The Latest Status of the CE Workgroup Shared Embedded Linux Distribution Project

Yoshitake Kobayashi
CE Workgroup, the Linux Foundation (Toshiba Corporation)
Embedded Linux Conference 2016
4-6 Apr, 2016

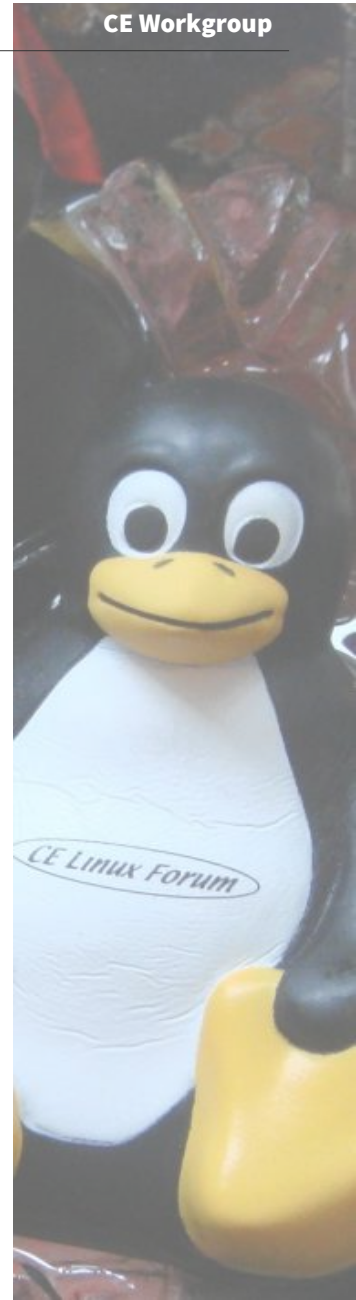
Overview

- **Introduction of CEWG**
- **The latest status of Shared Embedded Linux Distribution Project**

Role of CEWG / The Linux Foundation

- **Bridge** between OSS (like Linux) developer communities and embedded system industry who wish to collaborate with those communities.
  - Building the relation of trust and co-creation is so essential to realize the value of OSS
  - However it is so difficult achieving such ideal relationship solely by any company because of the diversified, groval and huge scale of the active OSS communities
- CEWG is a community of people who belong to the industry that wish to become a citizen of grater OSS communities and perform co-creation of innovative software

- For more information
  - CE Workgroup Linux Foundation
    http://www.linuxfoundation.org/collaborate/workgroups/celf



Activities of CEWG (CEWG Project)

- CEWG Project aim to study and to solve issues on embedded system

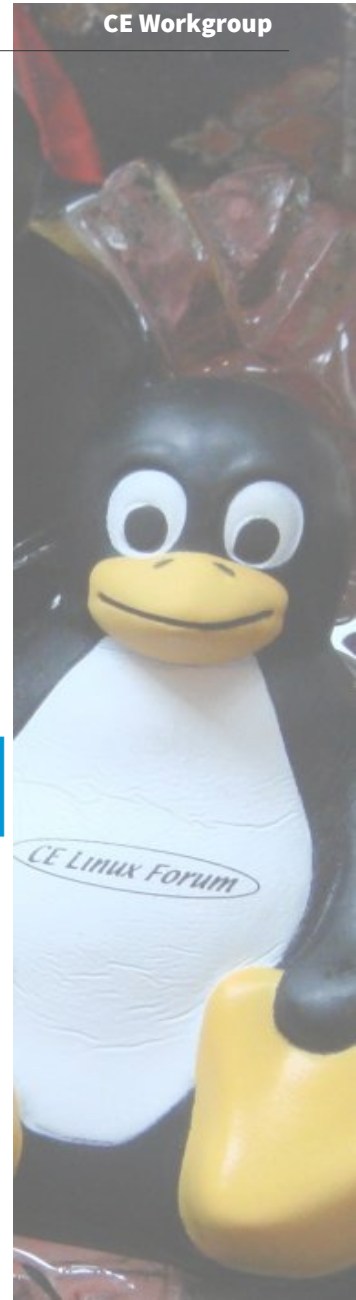
- Elinux.wiki
  - CEWG supports for the site
  - It is the technological information portal of OSS for embedded system developers

- Linux in Civil Infrastructure (was called Social Infrastructure)
  - Goals: Solve problems with Linux for use in civil infrastructure systems
  - Status: Launched a new collaborative project

- Device Mainlining
  - Goals: Study obstacles to mainlining, and work to reduce obstacles
  - Status: SIG meetings at ELCE and ELC
  - Presentations about overcoming obstacles at ELCE 2014, ELC 2015, and LCJ 2015
  - White paper (published at LCJ – June 2015)

- LTSI / LTSI-Testing
  - Goal: An extended support program based on industry requirement (Long Term Support Initiative) is carried out together with the Linux community
  - Status: Released LTSI-4.1 released. Fuego (aka. JTA) is presented at ELC2016

- Shared Embedded Distribution
  - Goals: Create an industry-supported distribution of embedded Linux and provide support for long term
  - Status: Created a Poky meta layer by using the Debian source code
  - Presented at conferences (ELC, LinuxCon)



Motivation

- **Linux is running on many kind of embedded systems**
  - Including the systems in civil infrastructure

- **Things to be considered to choose a base distribution**
  - The number of supported packages
  - Package versions
  - Supported hardware
  - Stability, number of bugs were fixed
  - The frequency of security updates and supported timespan
  - Easy to compile and to customize packages

In our use-case

- **What we want to do**
  - Make custom embedded Linux environments
- **What we need**
  - Wider hardware support
  - Stability
    - Well tested packages are required
    - Many embedded developer are still want to use stable version
  - Long-term support
    - Over 15 years support required, especially for security fixes
    - This is what we would like to contribute something
  - Fully customizable build system
  - Tools for production releases

Our solution

Yocto Project “Poky"

- One of the most popular reference distributions for embedded Linux
- Fully customizable build system
- Supports numerous embedded boards including modern ones
- Can be extended by meta-layer

Debian GNU/Linux

- Support many kind of CPUs: x86, ARM, PowerPC, MIPS (32bit/64bit)
- Release a stable version after two years of testing
- Long-term support for 5 years by Debian-LTS project

Create a meta layer by using Debian source (meta-debian)

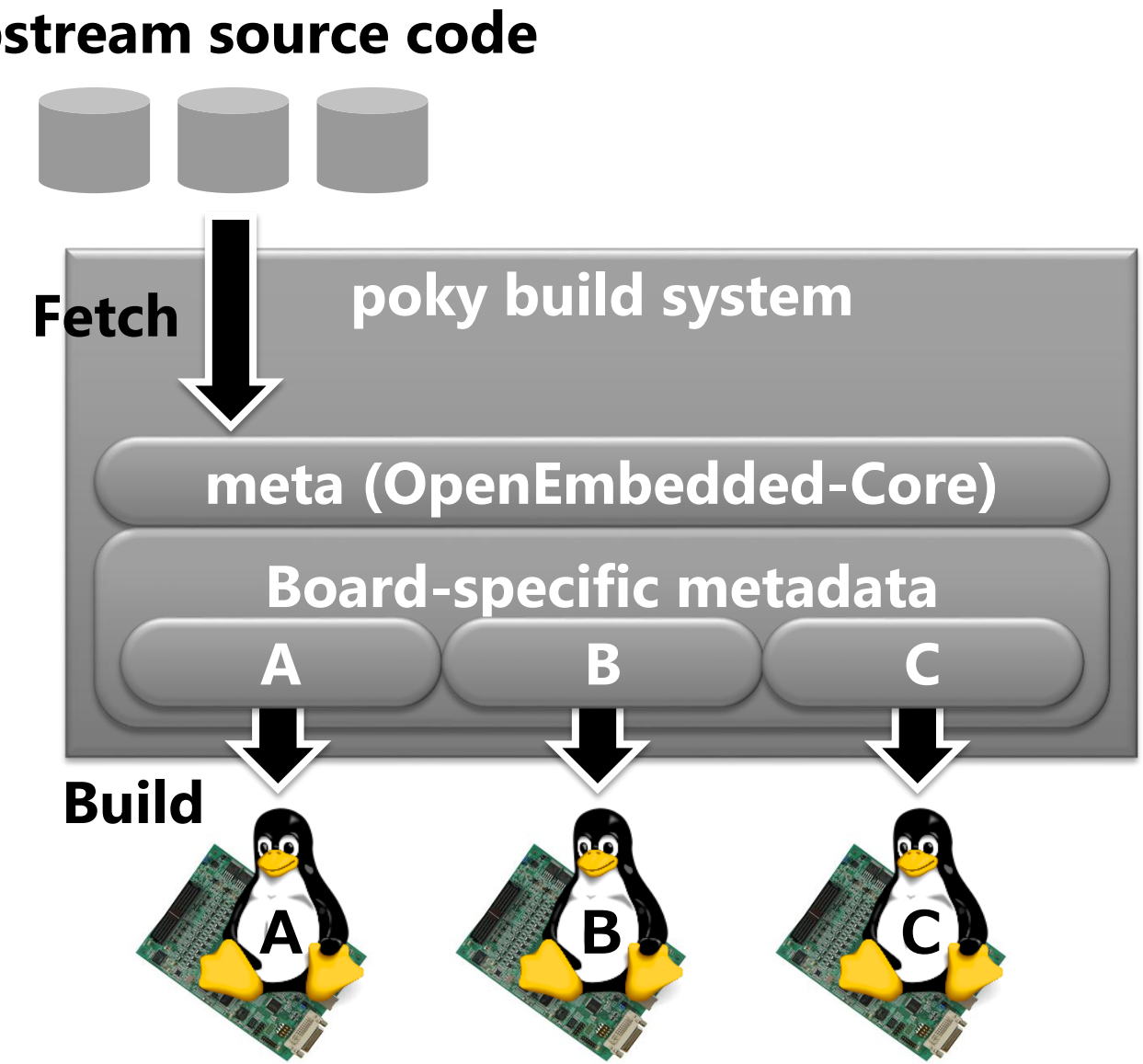
What is meta-debian layer?

- **A meta layer for the Poky build system**

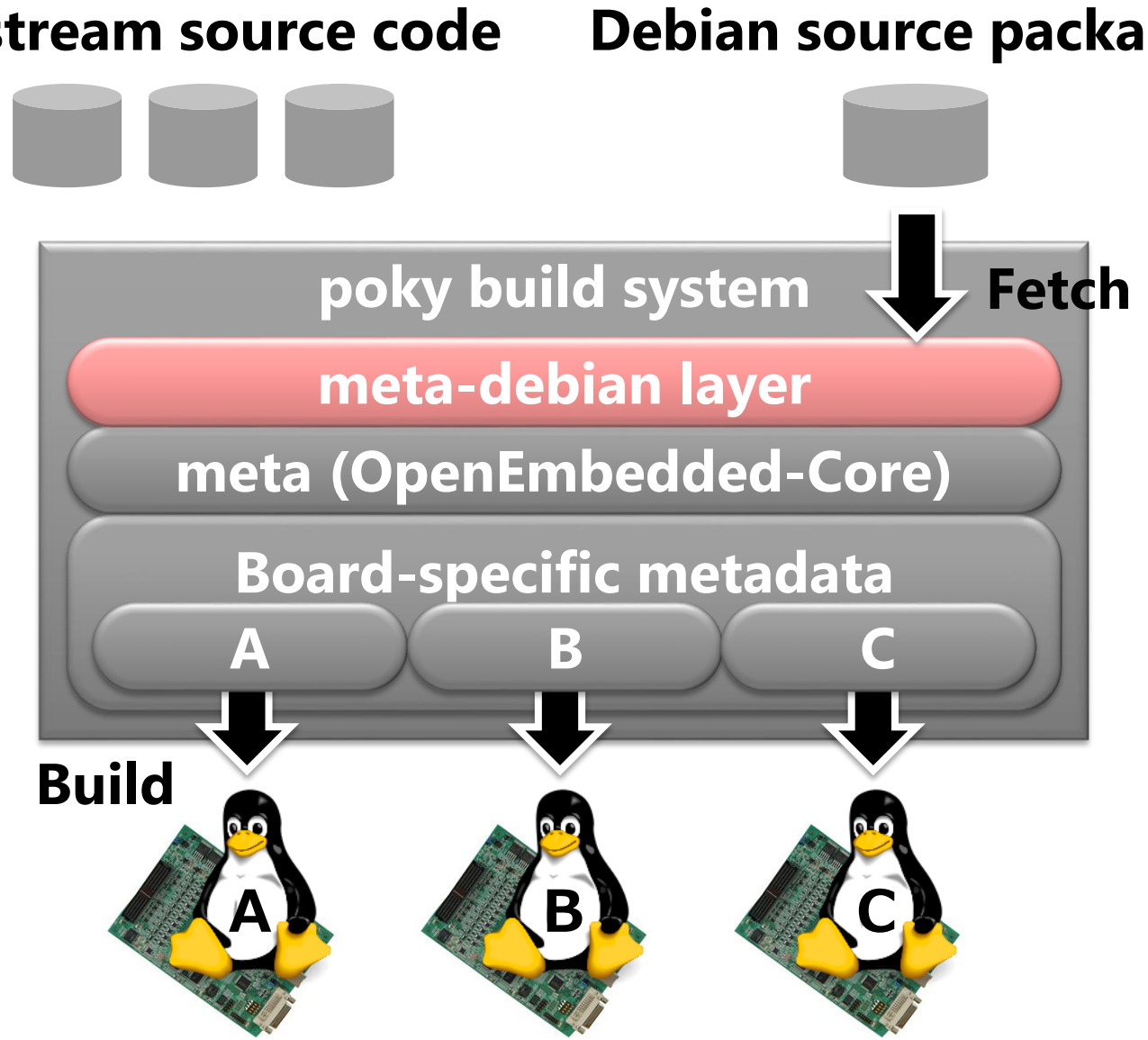
- **Main feature**
  - Cross-building Linux images by using Debian source codes

- **Implemented as an independent "layer"**
  - Separated from OpenEmbedded-Core and other layers
  - Source code: https://github.com/meta-debian/meta-debian.git

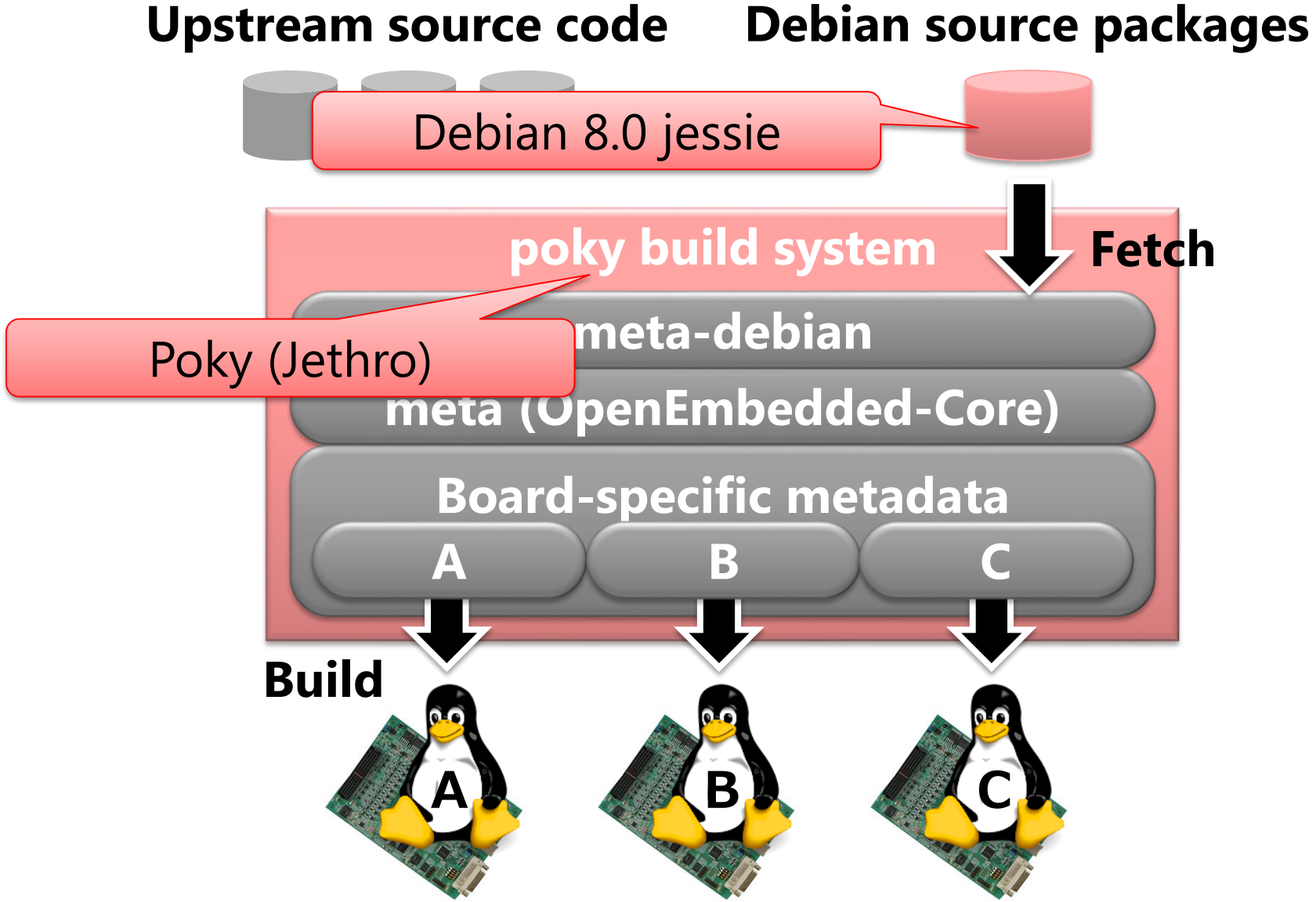
Build system structure (poky)



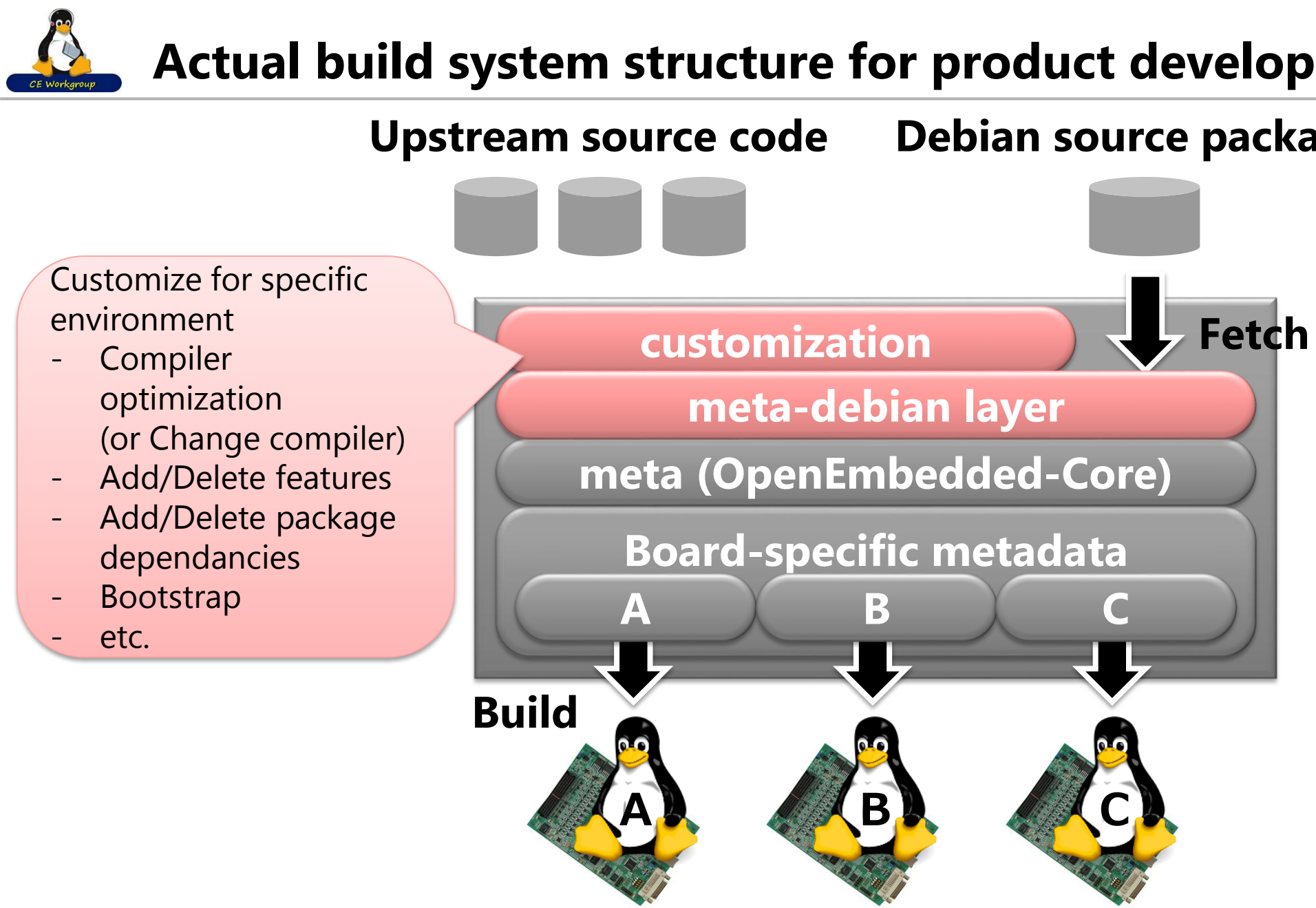
Build system structure (poky + meta-debian layer)



Target versions of meta-debian



Actual build system structure for product development



Purpose of meta-debian layer

- **Create embedded Linux environments with**
  - Wide embedded CPU support
  - Stability
  - Long-term support
  - Fully customizable build system

With Debian stable release + LTS

With poky build system

- **Provide a common place for developers having the same needs**
- **Contribute to relating project**
  - Debian
  - Yocto Project

Quick start

1. Download the build tools
2. Setup build directory
3. Build minimal Linux image
4. Run minimal Linux image on QEMU

- See also meta-debian/README

Download build tools

- **Download poky**

```
$ git clone git://git.yoctoproject.org/poky.git
$ cd poky
$ git checkout jethro
```

- **Download meta-debian layer into the poky directory**

```
$ cd poky
$ git clone https://github.com/meta-debian/meta-debian.git
$ cd meta-debian
$ git checkout jethro
```

- **← meta-debian layer specific step**

Setup build directory

- **Change the default configuration**
  - Enable meta-debian layer
  - Enable "debian" distro (DISTRO = "debian")
  - The default target machine is "qemux86" (MACHINE = "qemux86")
  - TEMPLATECONF is used by oe-init-build-env script

```
$ export TEMPLATECONF=meta-debian/conf
```

- **Run startup script**
  - This setup a build directory and environment variables automatically
  - (builddir): name of build directory (optional)

```
$ source /path/to/poky/oe-init-build-env (builddir)
```

Build minimal Linux image

- **Run bitbake**

```
$ bitbake core-image-minimal
```

- **Built images (case of qemux86)**
  - Output directly
    - /path/to/builddir/tmp/deploy/images/qemux86
  - Kernel
    - bzImage-qemux86.bin
  - Root filesystem
    - core-image-minimal-qemux86.ext3
    - core-image-minimal-qemux86.tar.gz

Run minimal Linux image on QEMU

- **Run built images on QEMU environment**
  - qemux86

```
$ runqemu qemux86 nographic bootparams="init=/init root=/dev/sda"
```

  - qemux86-64

```
$ runqemu qemux86-64 nographic bootparams="init=/init root=/dev/sda"
```

  - qemuarm

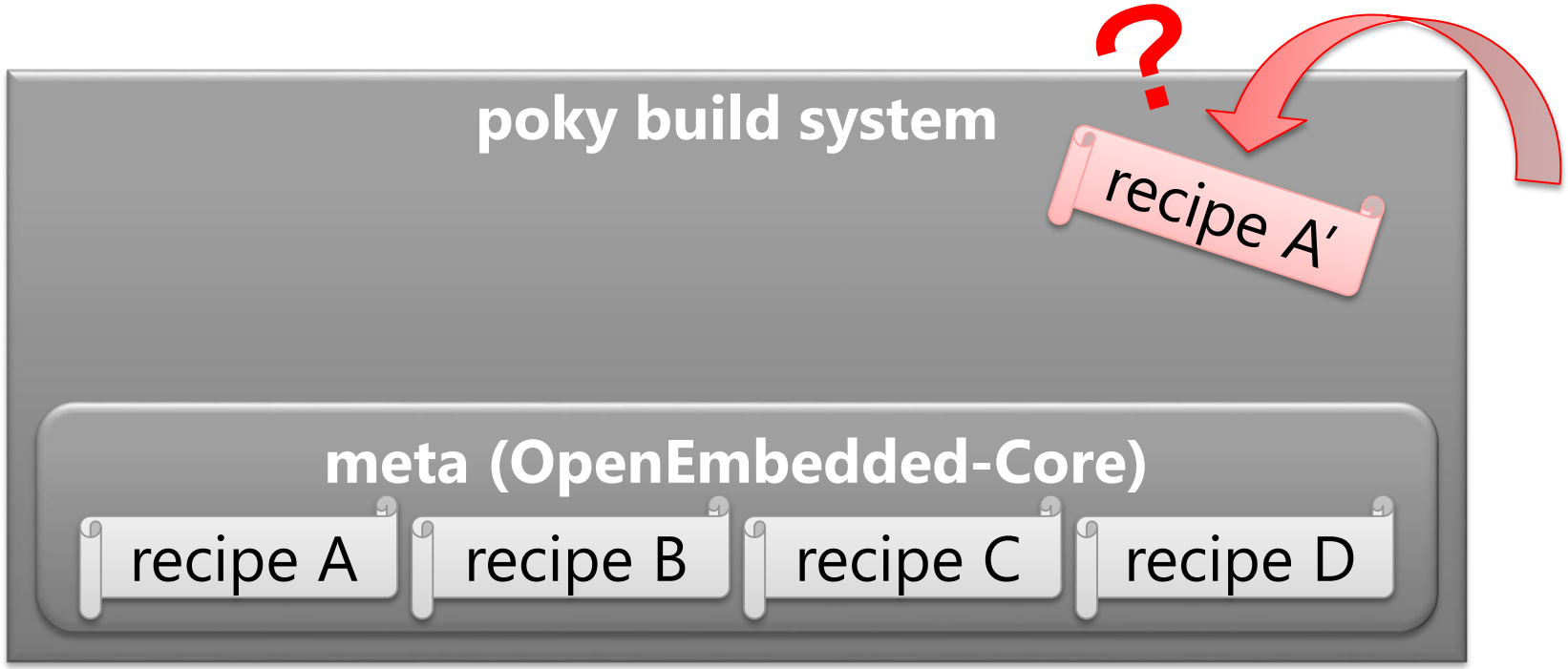
```
$ runqemu qemuarm nographic bootparams="init=/init console=ttyAMA0"
```

  - qemuppc

```
$ runqemu qemuppc nographic bootparams="init=/init"
```

How should we create recipe files?

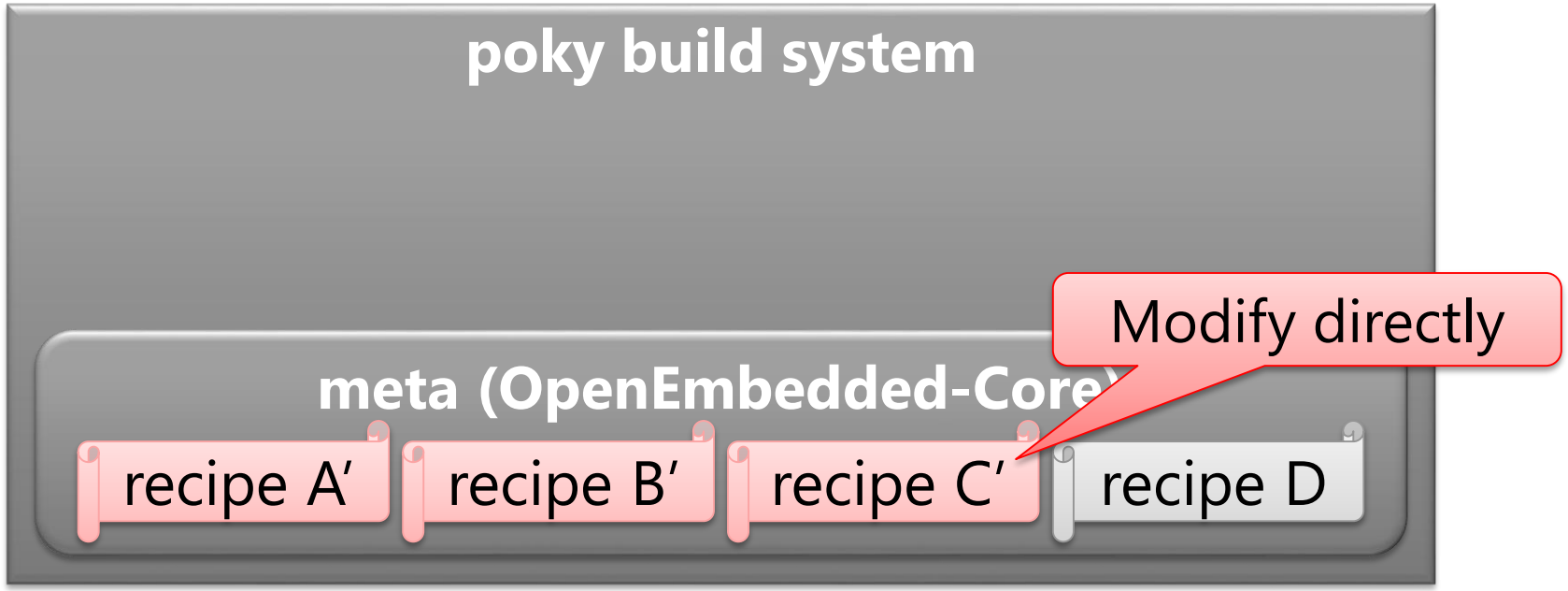
- **We need to create new recipes for Debian sources**
  - How?



- **Possible solutions**
  - Method 1: Modify OE-Core recipes directly
  - Method 2: Add recipes into a new layer

Method 1: Modify OE-Core recipes

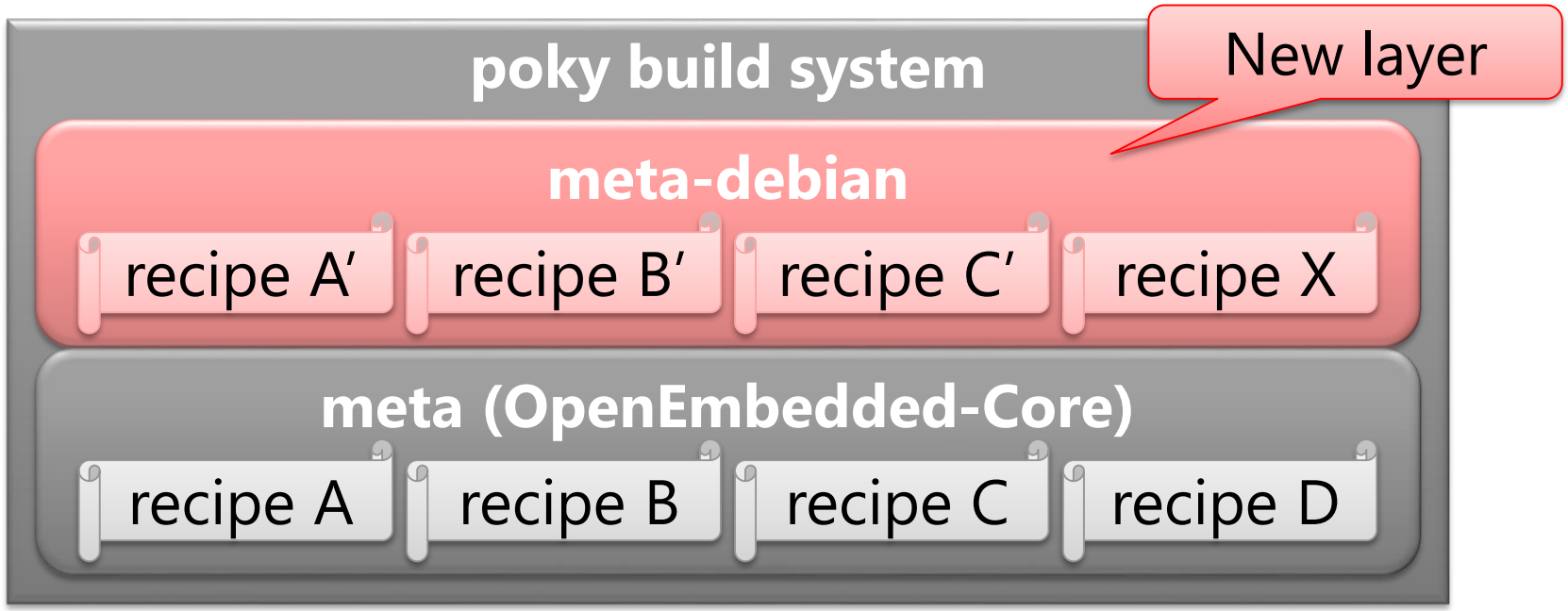
- **We already tried this way previously… ,but**
- **Not the ideal solution ☹**
  - Original OE-Core recipes are no longer available (Modified)
  - Just a fork
    - It becomes hard to catch up with the newest poky versions
    - Difficult to convince other people to join our effort



Method 2: Add recipes into a new layer

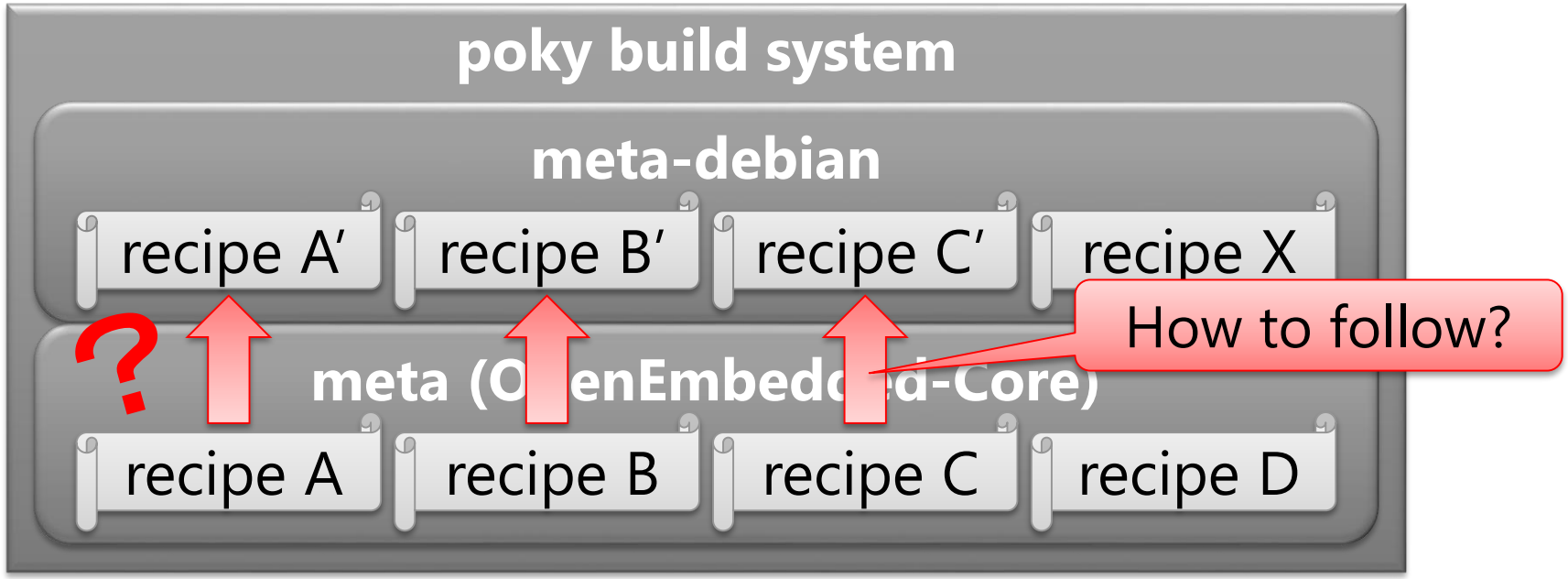
our solution

- **The best way to add new recipes for specific purposes**
  - Original OE-Core recipes are still there
  - Can be developed independently of OE-Core
  - Enable / disable the layer easily like a module



How should we create recipes in a layer?

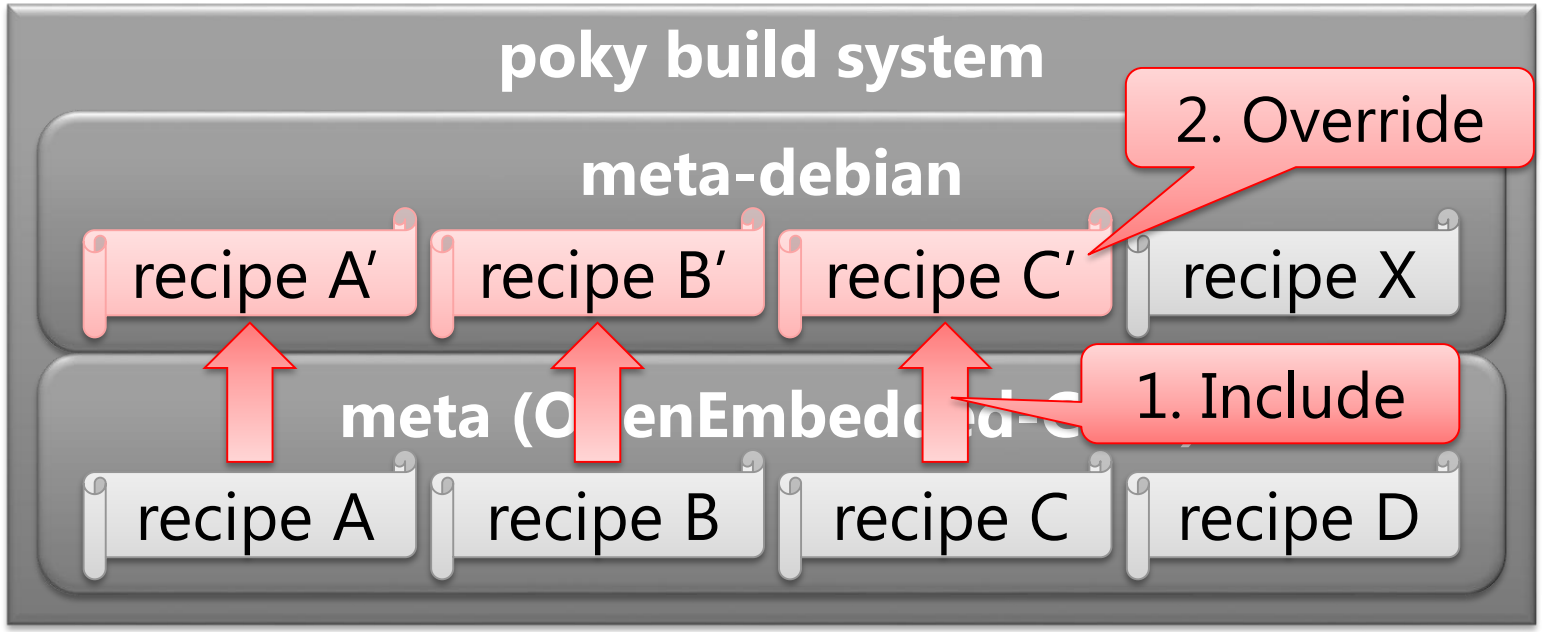
- **From scratch?**
  - Often takes time!
  - Why?
    - Need to create patches for supporting cross-build in poky
- **We should follow the existing OE-Core recipes**
  - How?



- **Method 1: "Include" OE-Core recipes**
- **Method 2: Use a part of OE-Core recipes**

Method 1: "Include" OE-Core recipes

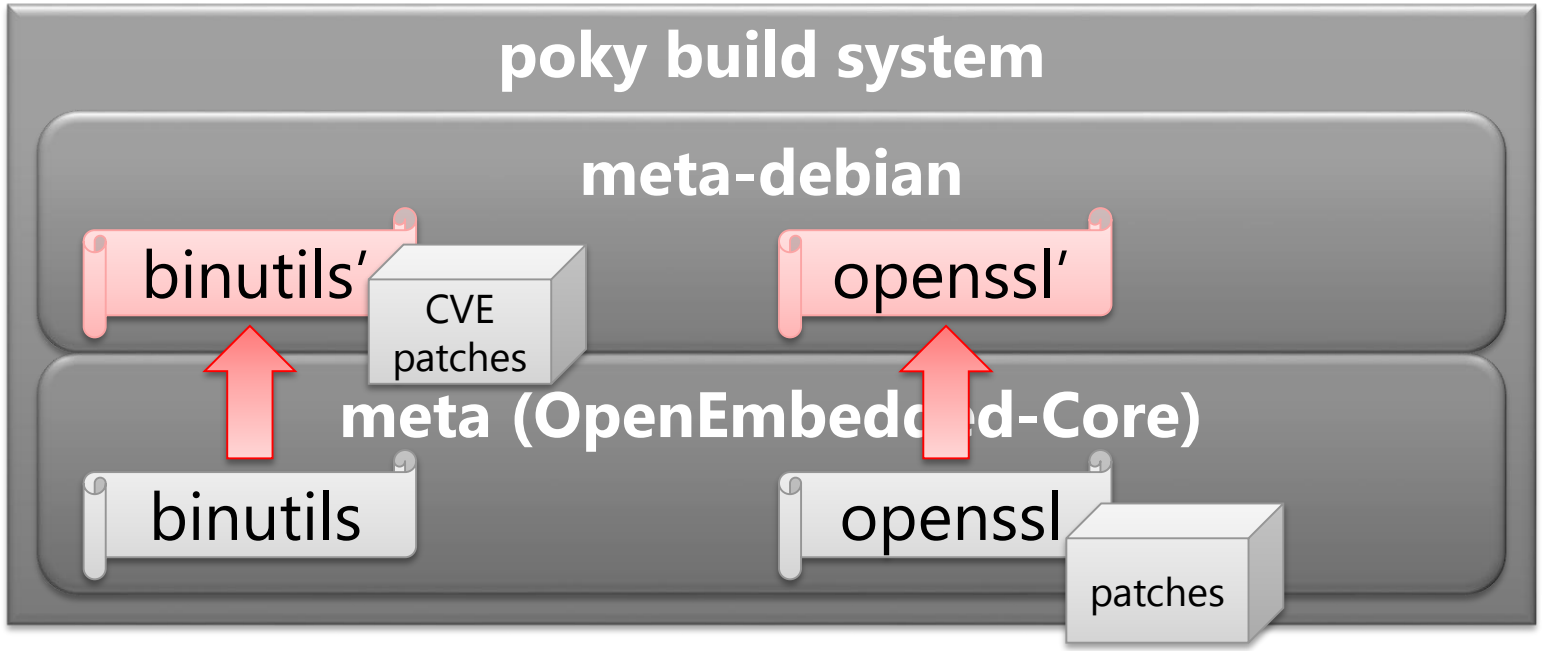
- **We used to use this method before…**
- **Unsuitable for our case ☹**
  - Difficult to override some variables and functions
    - Ex: already appended (_append) or prepended (_prepend) data
  - Automatically follow "unneeded" OE-Core updates against our will
    - Ex: Shown in the next slides



Method 1: "Include" OE-Core recipes

- **binutils**

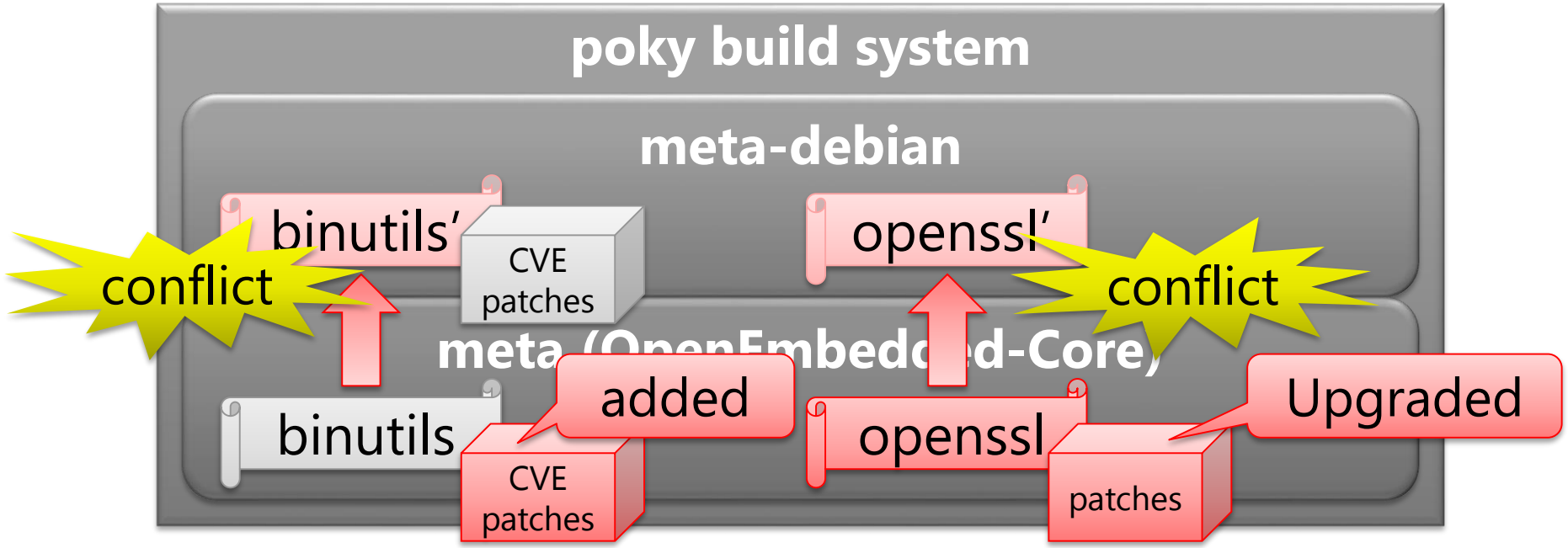
- **openssl**



Method 1: "Include" OE-Core recipes

- **binutils**
  - Security patches applied twice
- **openssl**
  - Target version was upgraded, and patches also upgraded
  - Some upgraded patches conflict with Debian source

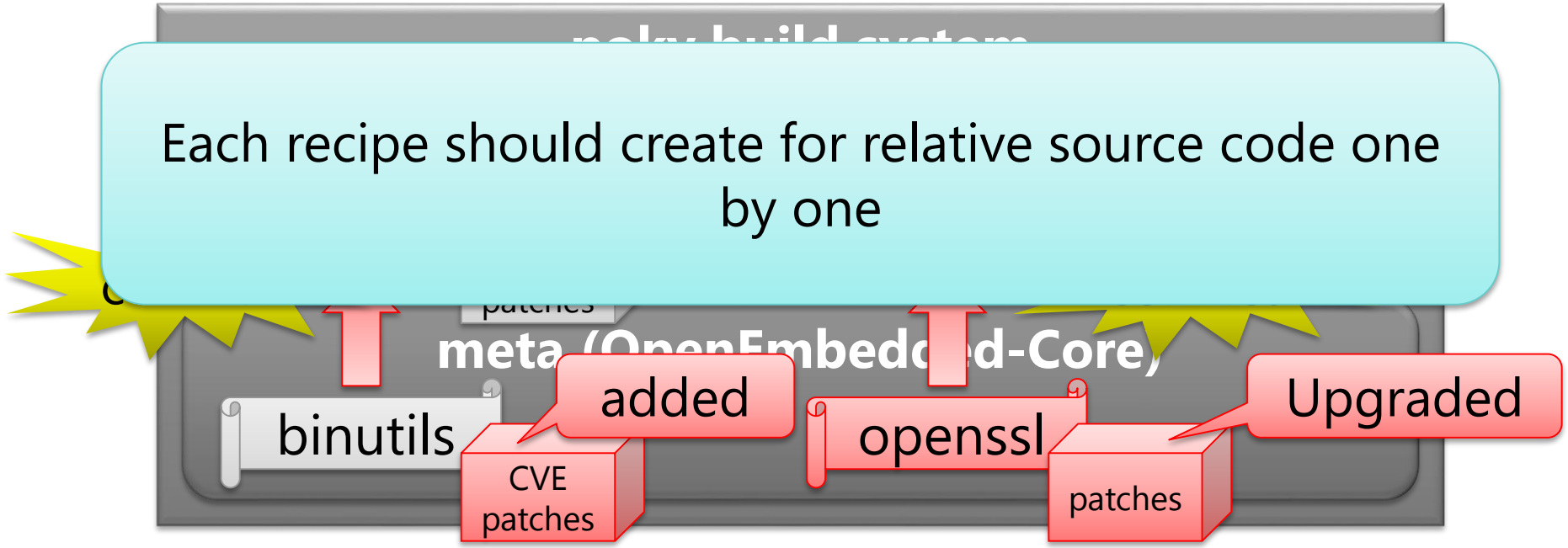
Difficult to maintain ☹



Method 1: "Include" OE-Core recipes

- **binutils**
  - Security patches applied twice
- **openssl**
  - Target version was upgraded
  - Some upgraded patches conflict with Debian source

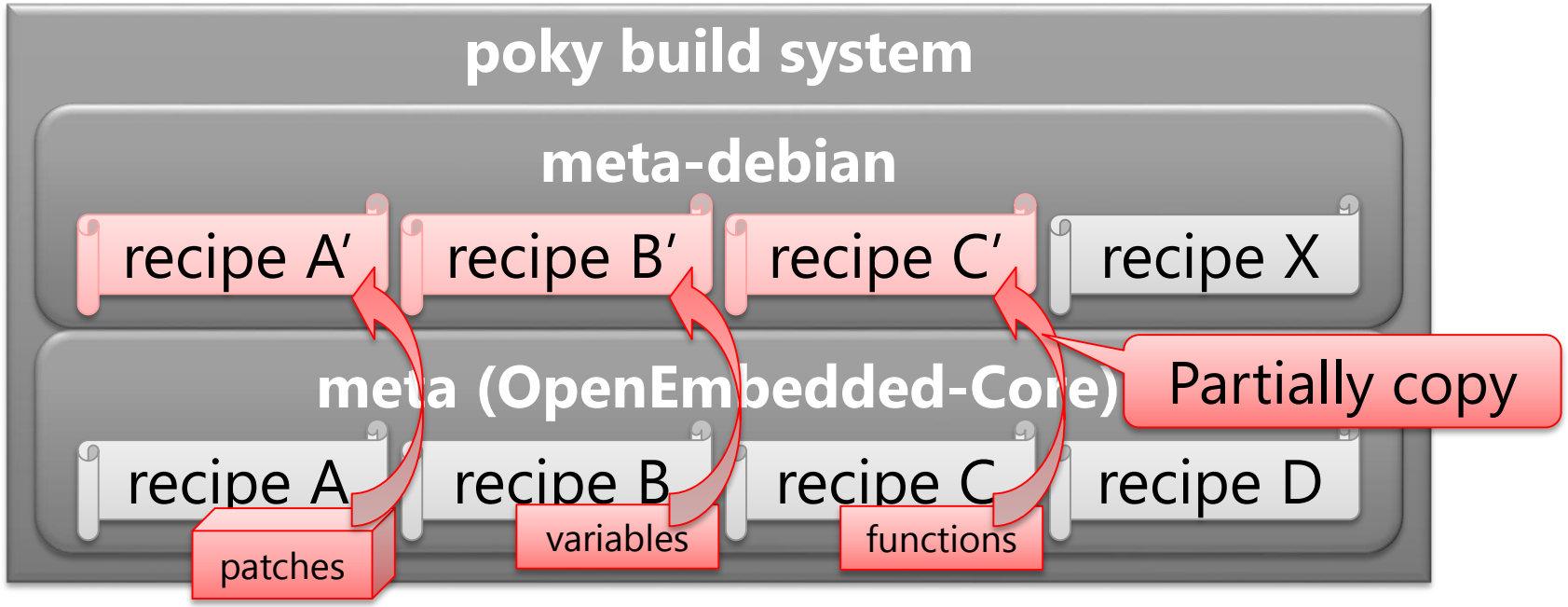
Difficult to maintain ☹



Method 2: Use a part of OE-Core recipes

our solution

- **Try to create recipes from scratch using Debian source packages**
- **Re-use the essential data from OE-Core**
  - patches, variables, functions for supporting cross-build



How should we implement recipes?

- LICENSE information
- Required files
  - Source code
  - initscripts, configs
  - Patches
- Configure commands & options
- Compile commands & options
- How to installed files
- How to make the package file
- Package dependencies

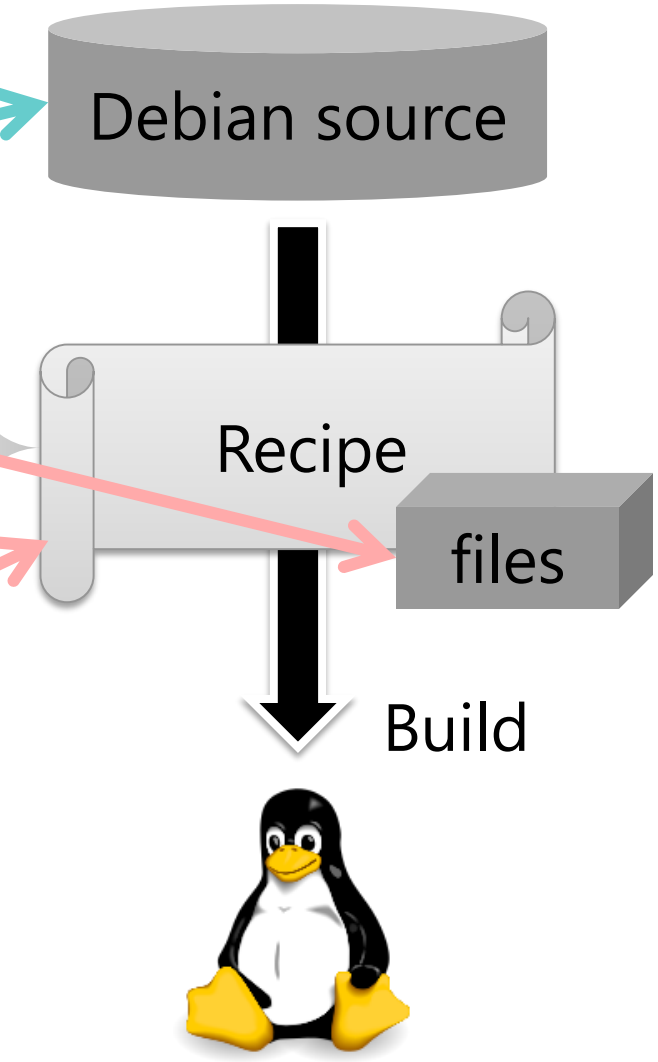
- **The following slide describes:**
  - Method 1: just re-use OE-Core recipes
  - Method 2: Follow Debian's packaging

Method 1: re-use OE-Core (not a good solution)

- LICENSE information
- Required files
  - Source code
  - initscripts, configs
  - Patches
- Configure commands & options
- Compile commands & options
- Installed files and paths
- How to package files
- Dependencies between others

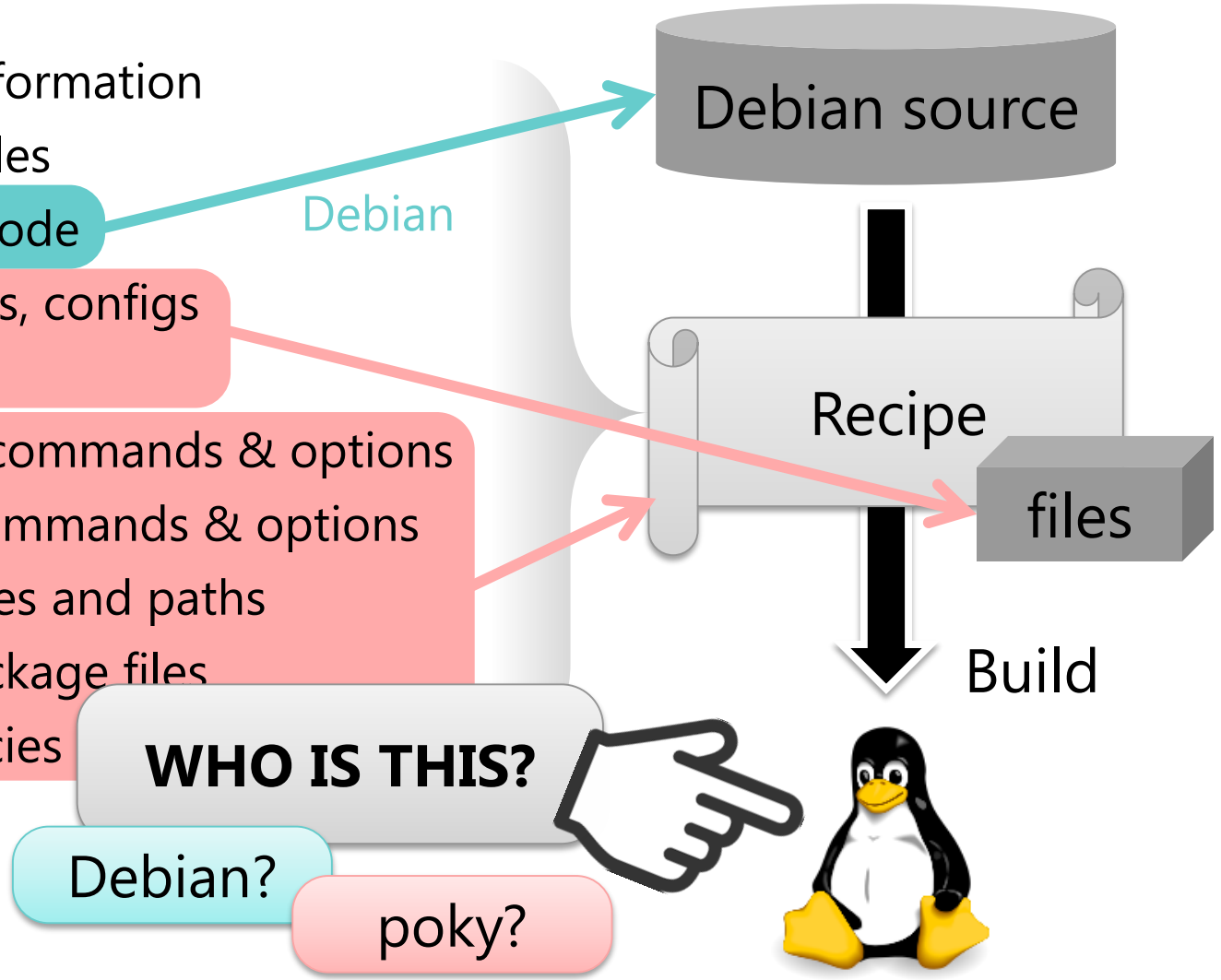
Debian

OE-Core based



Method 1: re-use OE-Core (not a good solution)

- LICENSE information
- Required files
  - Source code
  - initscripts, configs
  - Patches
- Configure commands & options
- Compile commands & options
- Installed files and paths
- How to package files
- Dependencies



Method 1: re-use OE-Core (not a good solution)

- **Bad results: conflicts of two distributions**
  - Compile fails
    - Cause: missing configure options that Debian source requires
  - Some programs fail to call commands or load data file
    - Cause: installation paths differ from Debian's
- **Cannot be used like Debian**

We should define some development "policy" for creating recipes

Policies for creating recipes

our solution

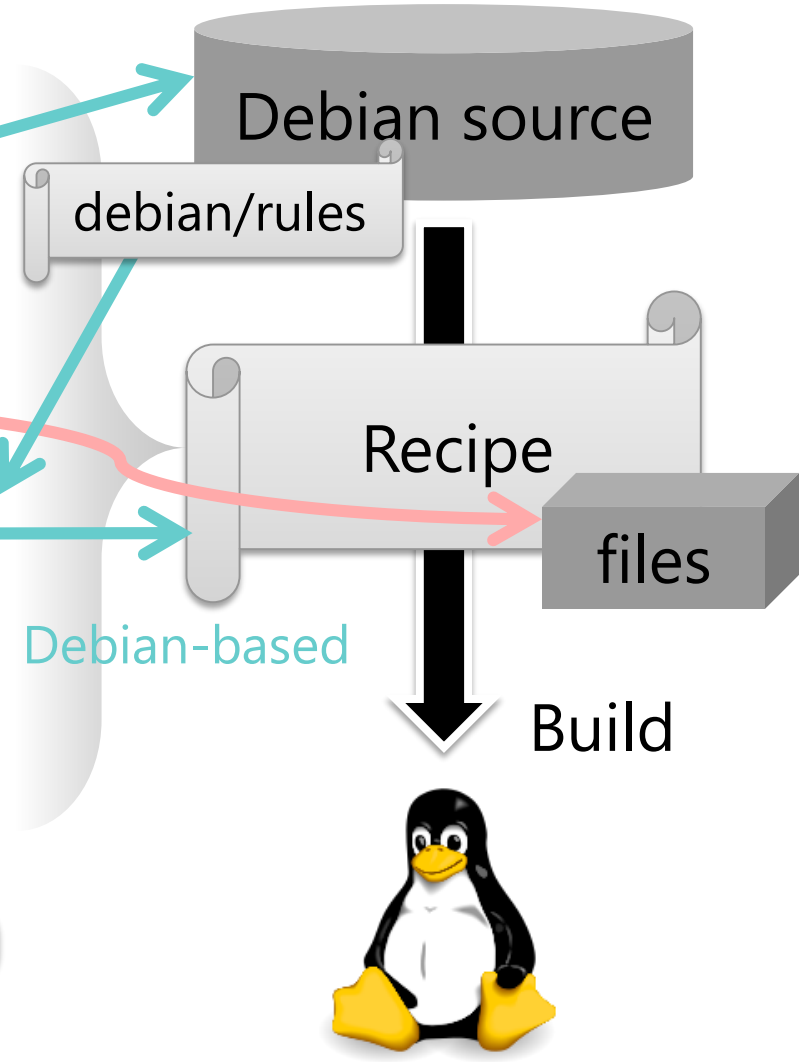
- **By default, follow Debian's packaging**
  - i.e. debian/rules
  - For getting good affinity with Debian sources
- **Customize for embedded system if necessary**
  - Disable features
  - Remove dependencies
- **Re-use only essential data from OE-Core for supporting cross-compile**
  - See "Method 2: Re-use OE-Core recipes"

Method 2: Follow Debian's packaging

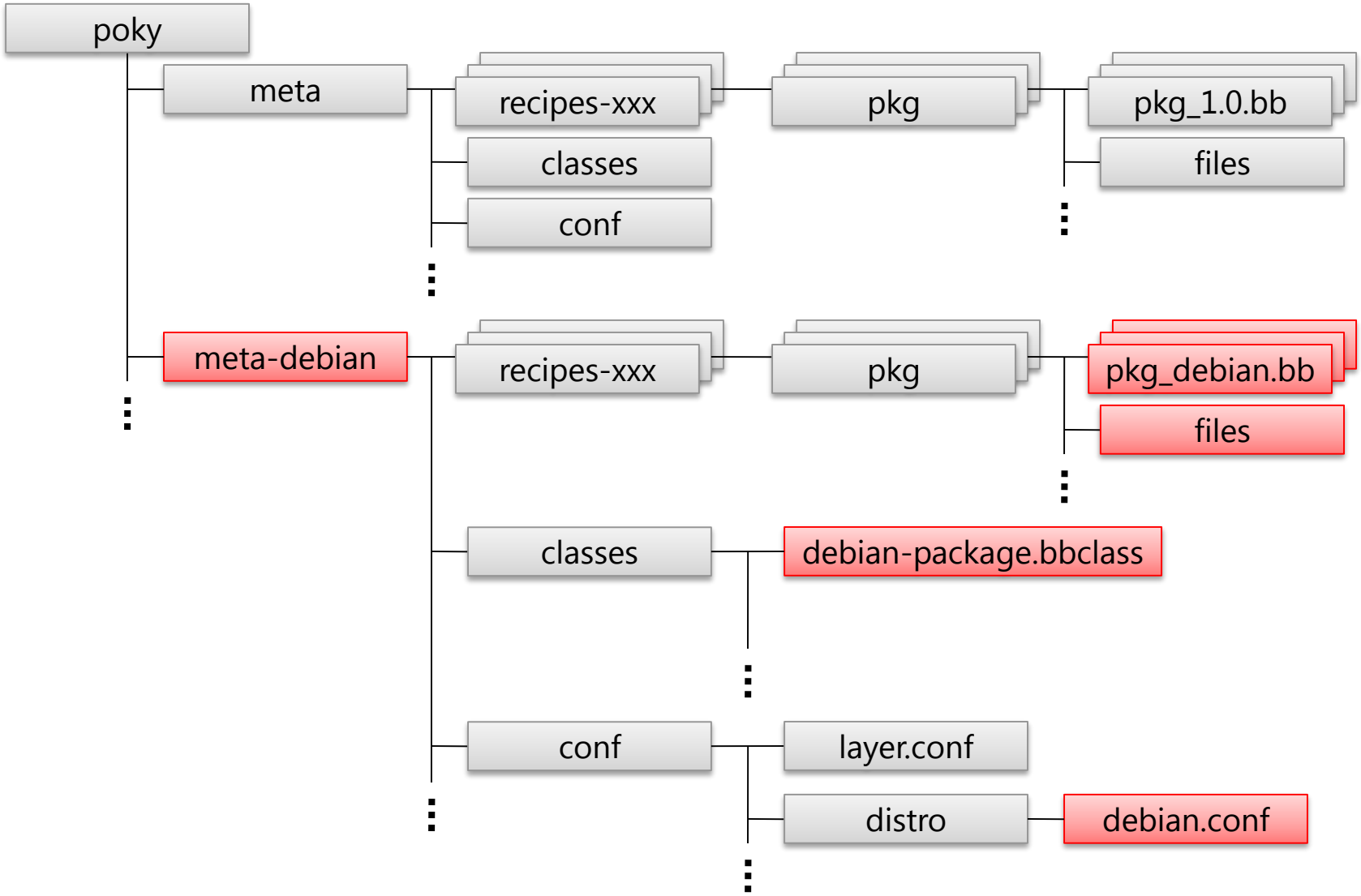
our solution

- LICENSE information
- Required files
  - Source code
  - initscripts, configs
  - Patches
- Configure commands & options
- Compile commands & options
- Installed files and paths
- How to package files
- Dependencies between others

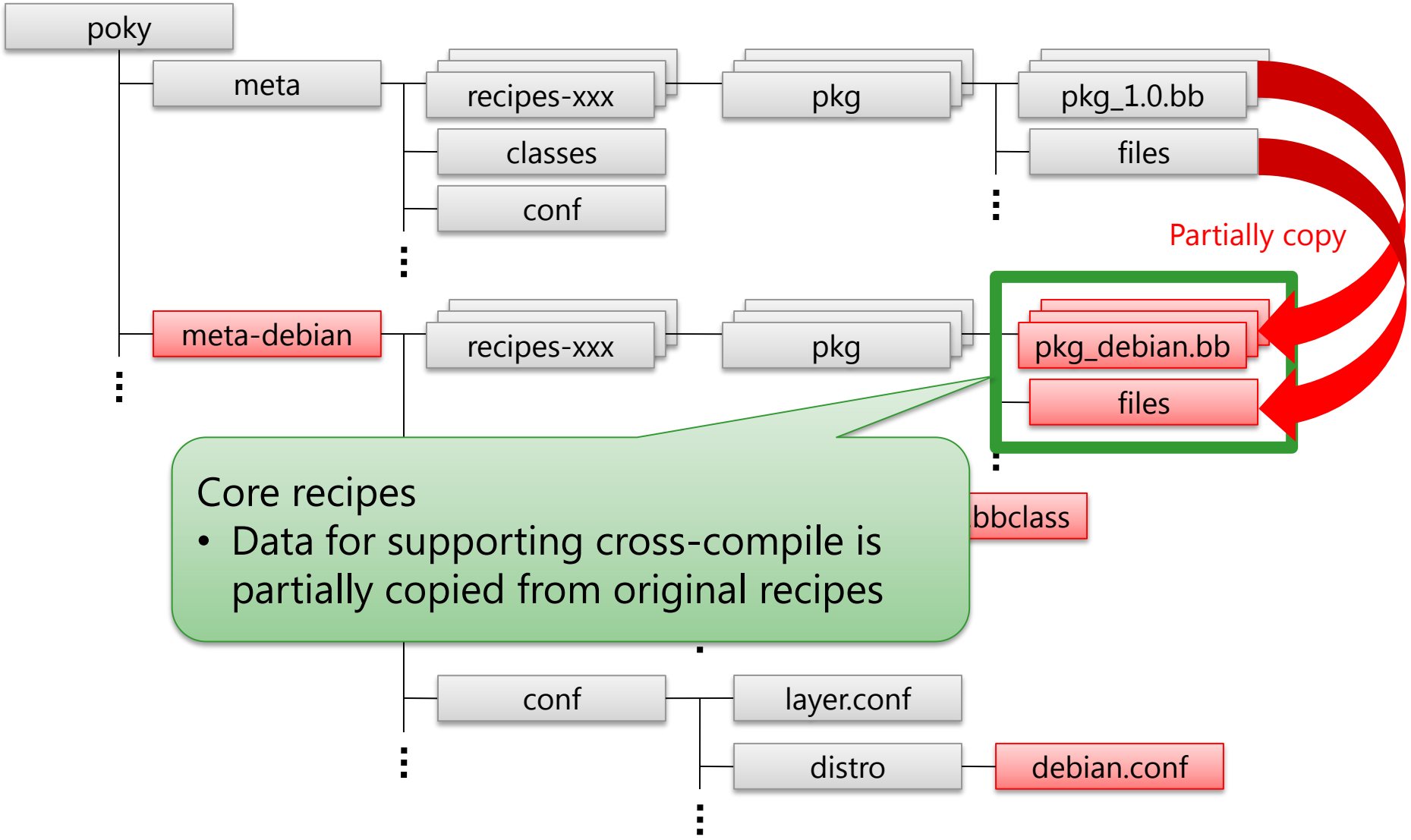
Customize for embedded



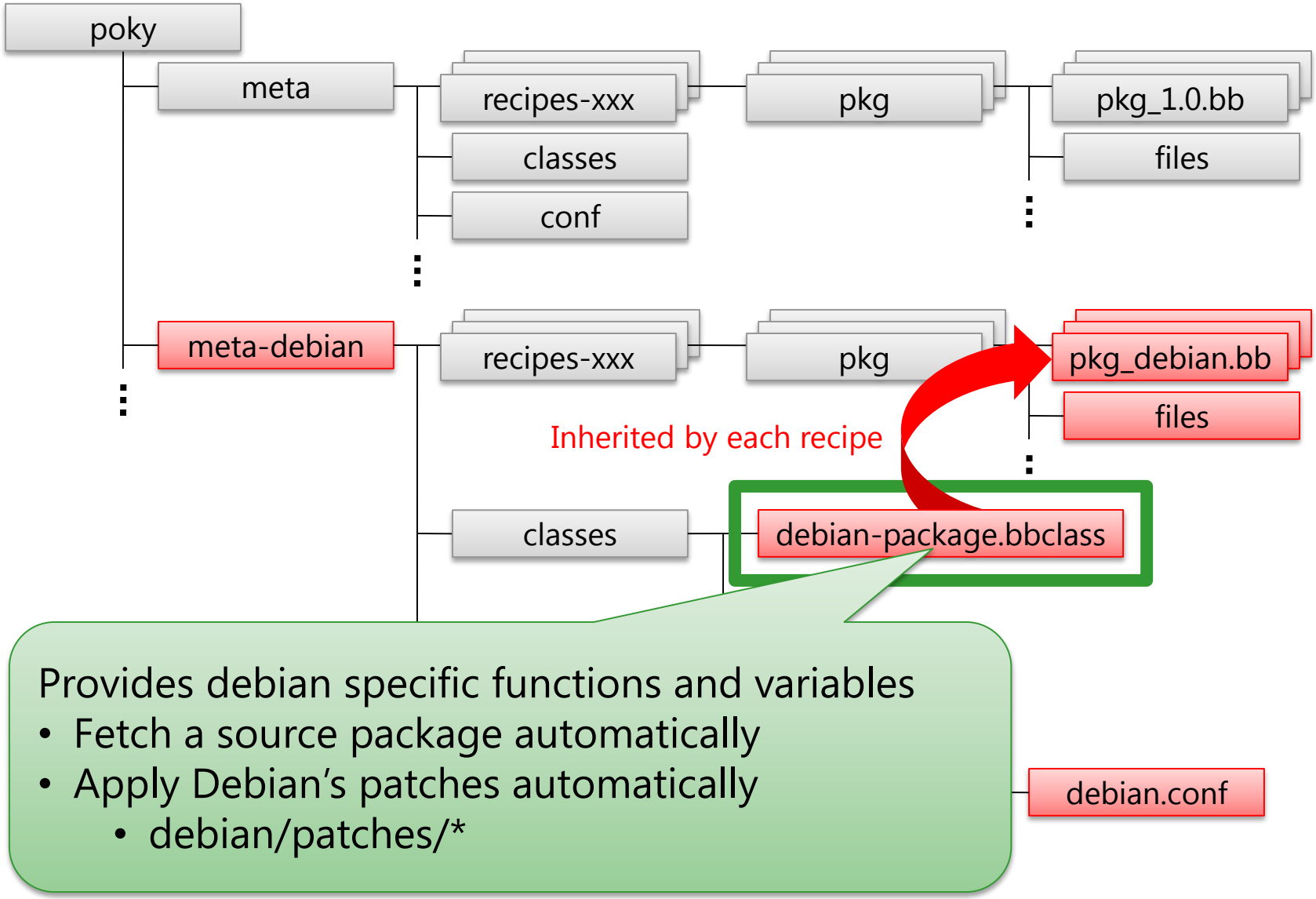
Directory structure



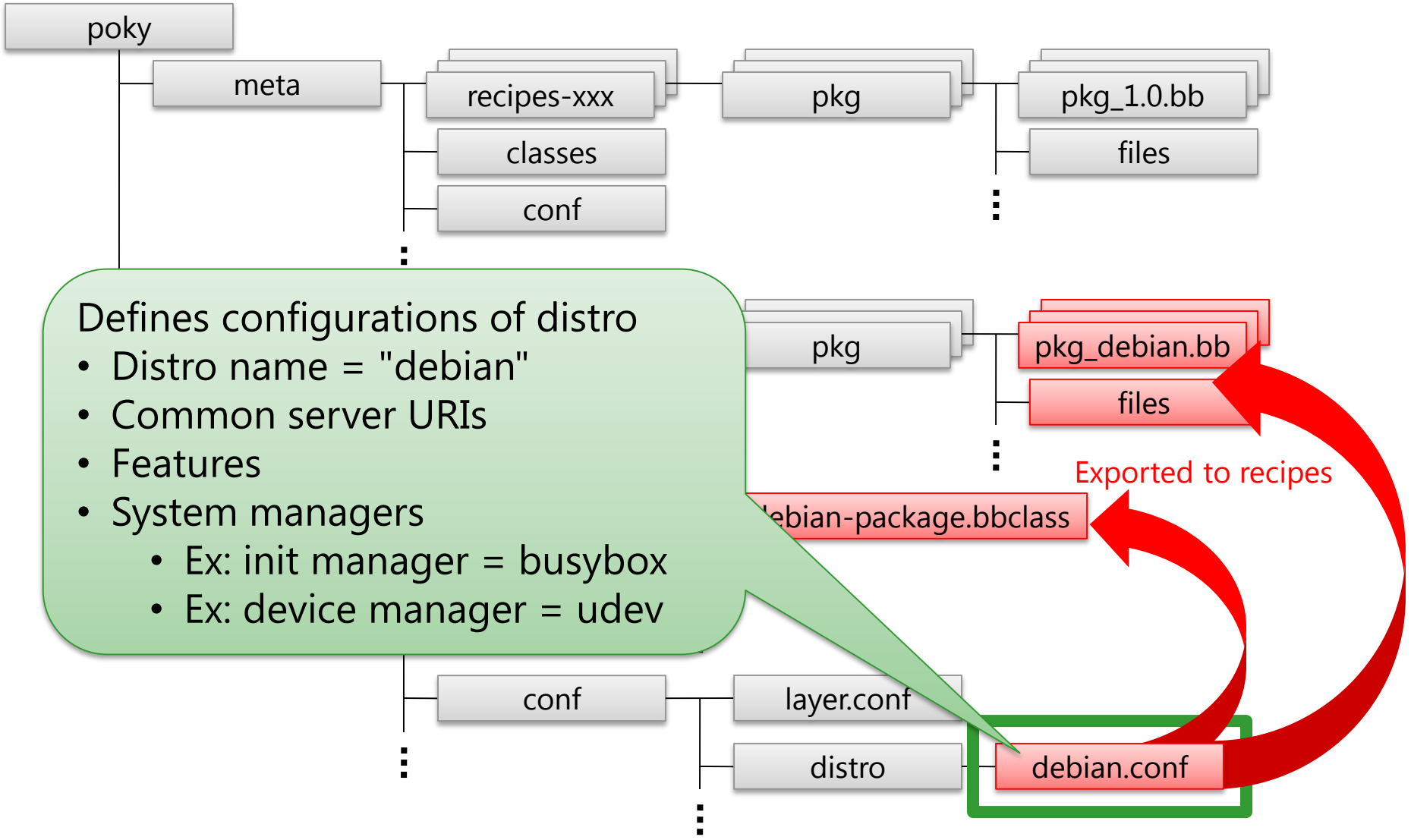
Directory structure



Directory structure



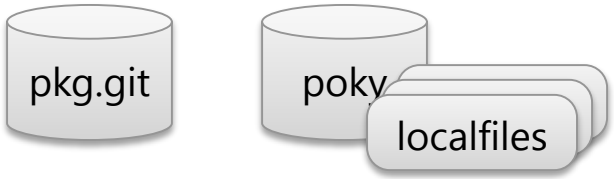
Directory structure



Build flow

bitbake tasks

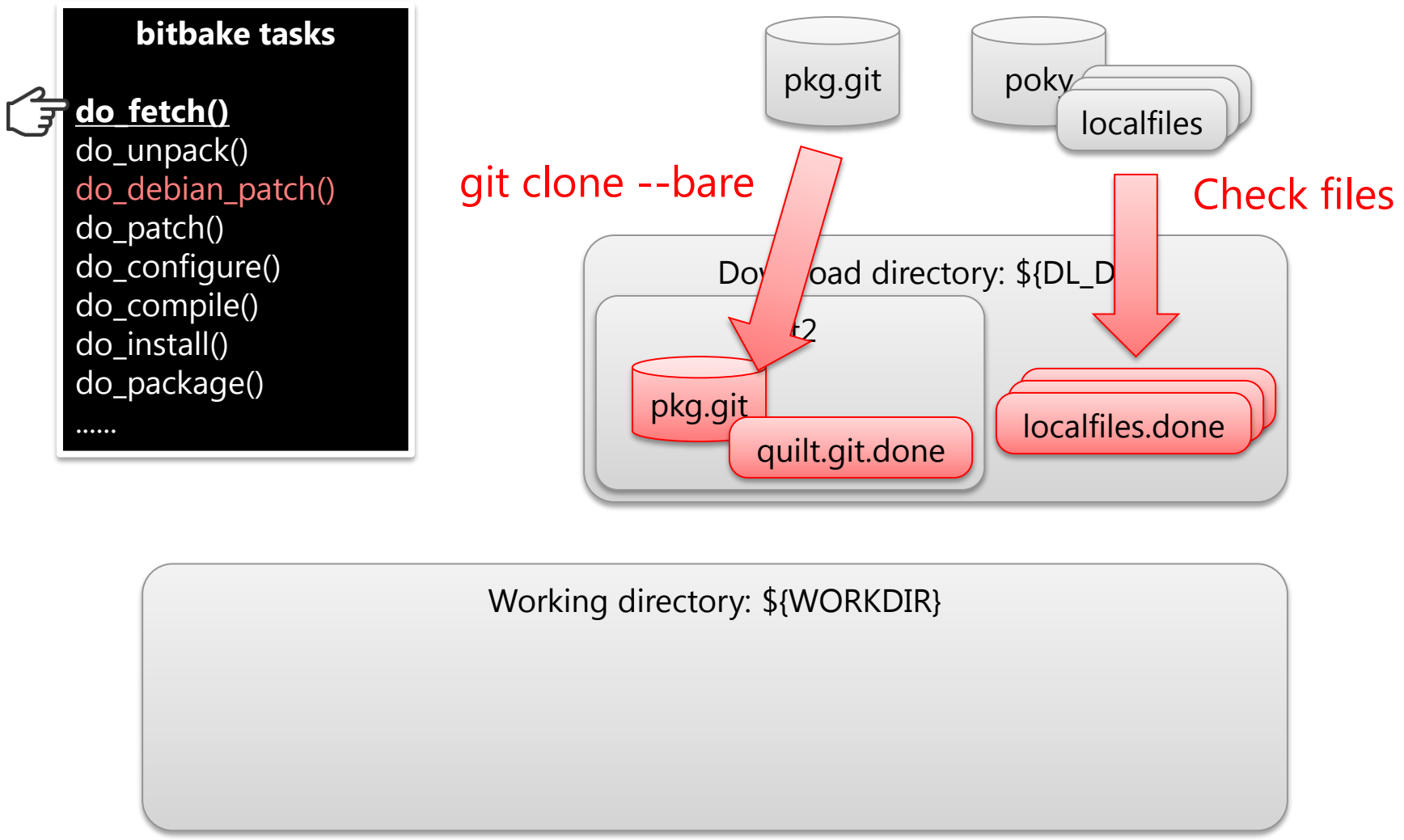
do_fetch()
do_unpack()
do_debian_patch()
do_patch()
do_configure()
do_compile()
do_install()
do_package()
.....



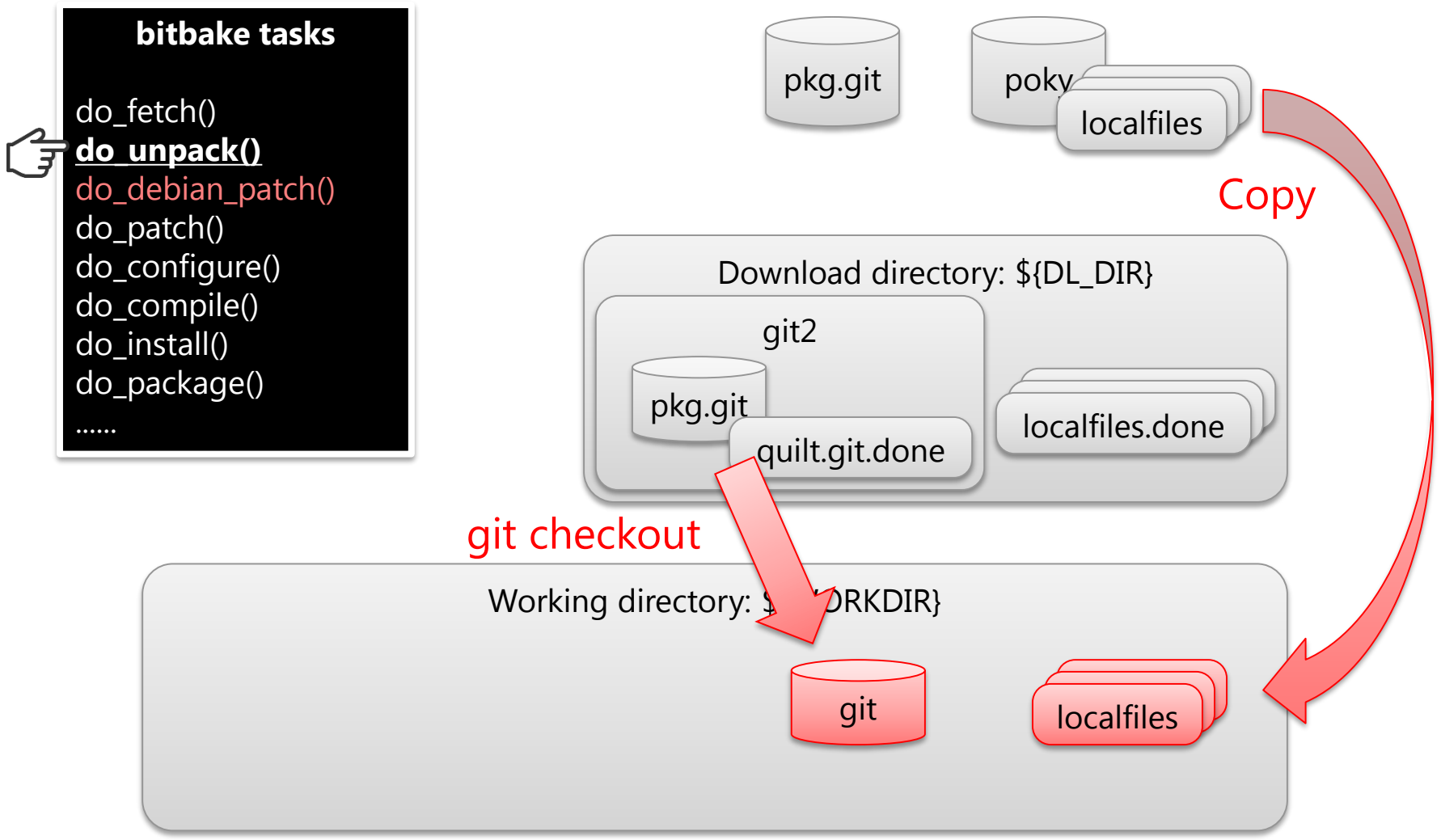
Download directory: ${DL_DIR}

Working directory: ${WORKDIR}

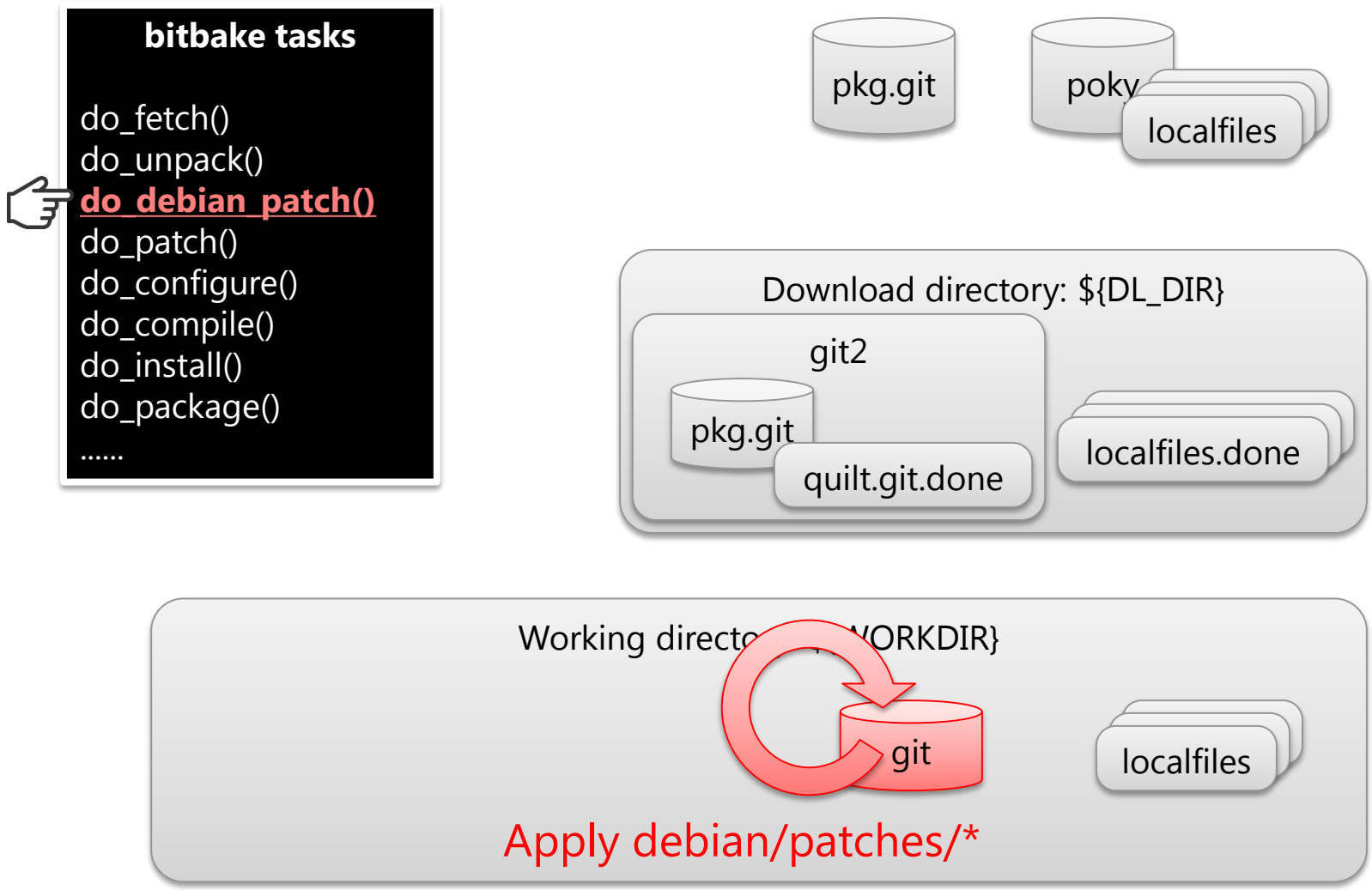
Build flow


bitbake tasks
do_fetch()
do_unpack()
do_debian_patch()
do_patch()
do_configure()
do_compile()
do_install()
do_package()
.....
pkg.git
poky
localfiles
git clone --bare
Check files
pkg.git
quilt.git.done
localfiles.done
Working directory: ${WORKDIR}

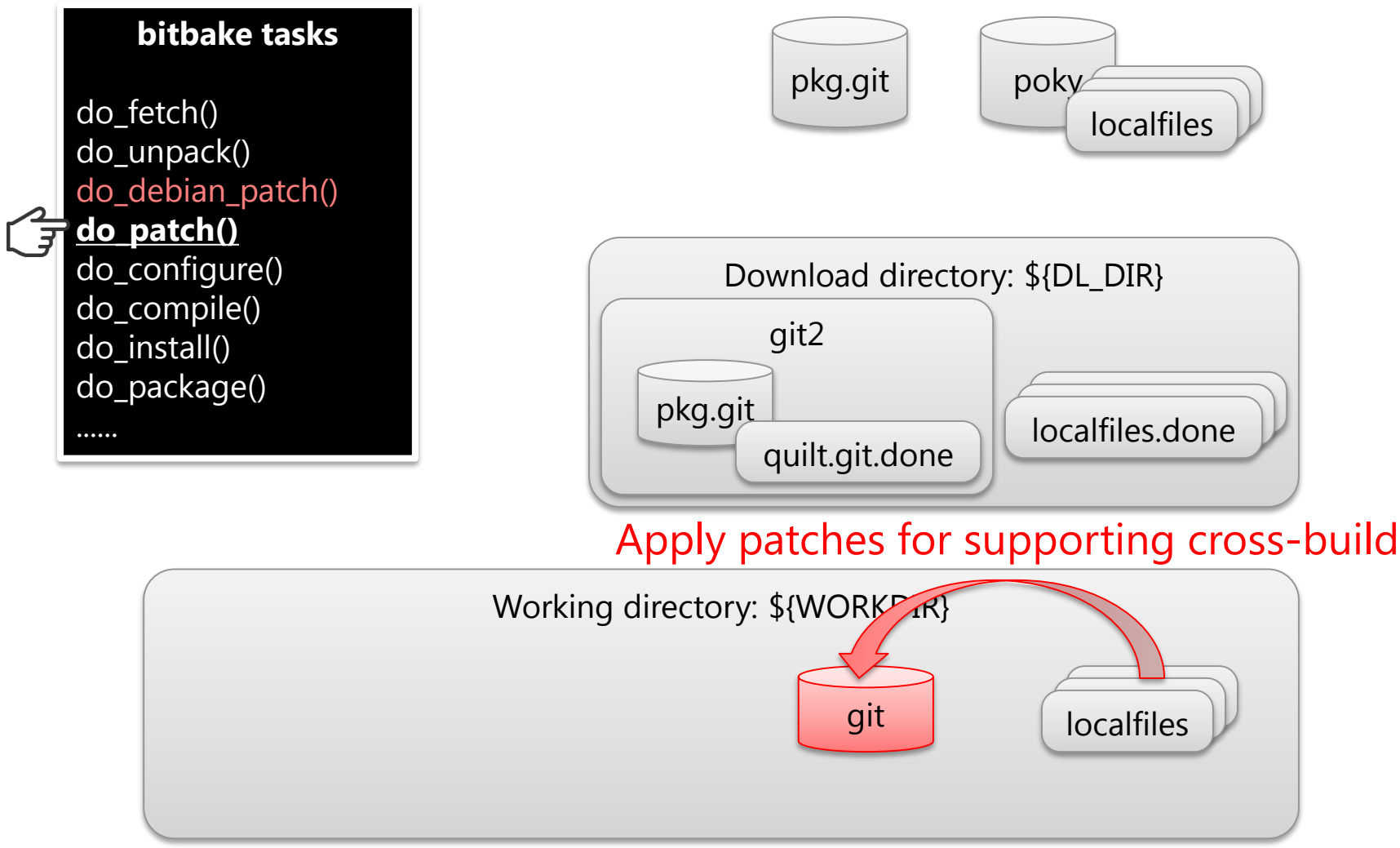
Build flow


bitbake tasks
do_fetch()
do_unpack()
do_debian_patch()
do_patch()
do_configure()
do_compile()
do_install()
do_package()
.....
pkg.git
poky
localfiles
Copy
Download directory: ${DL_DIR}
git2
pkg.git
quilt.git.done
localfiles.done
git checkout
Working directory: ${WORKDIR}
git
localfiles

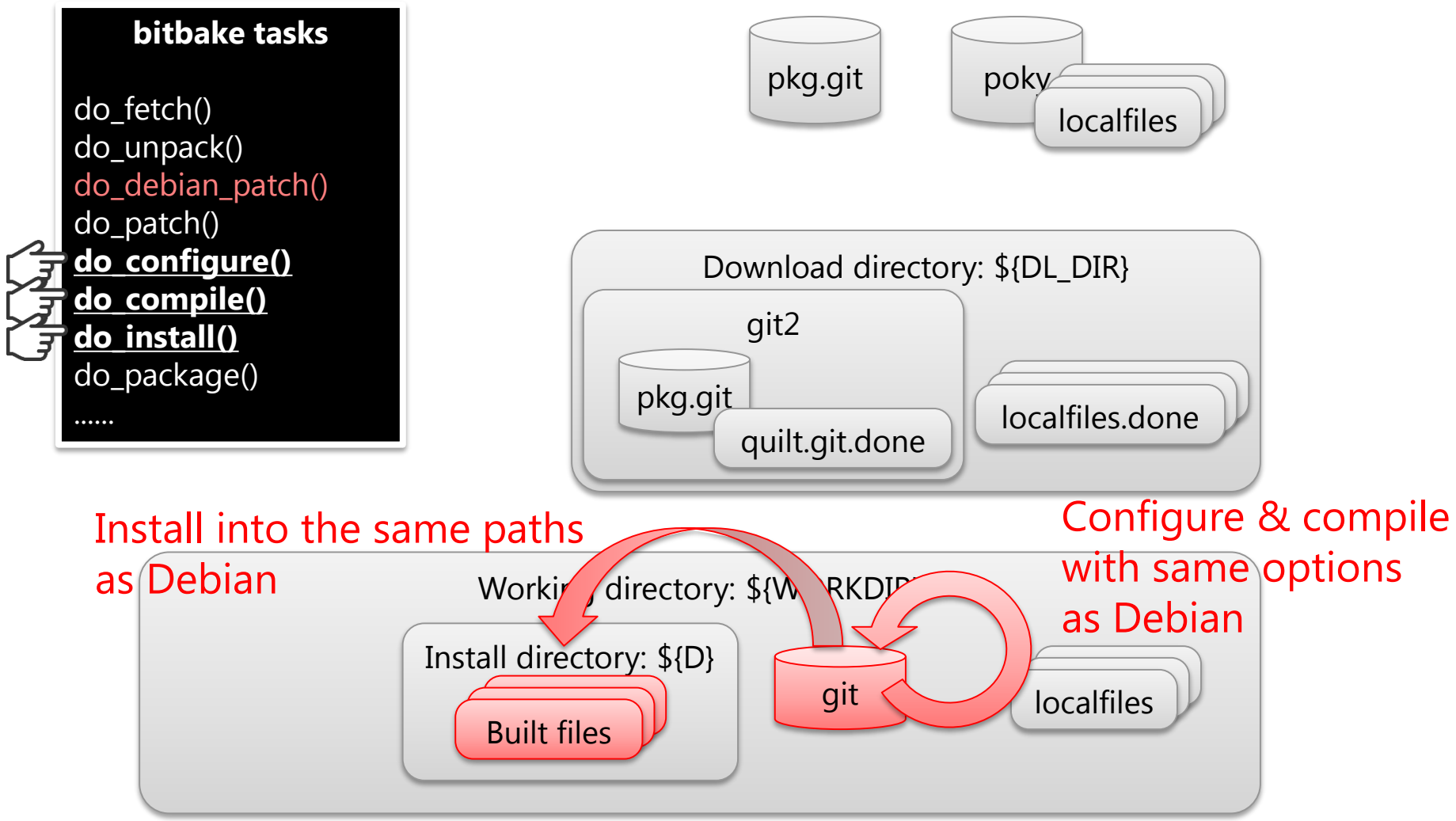
Build flow


bitbake tasks
do_fetch()
do_unpack()
do_debian_patch()
do_patch()
do_configure()
do_compile()
do_install()
do_package()
.....
pkg.git
poky
localfiles
Download directory: ${DL_DIR}
git2
pkg.git
quilt.git.done
localfiles.done
Working directo ORKDIR}
git
localfiles
Apply debian/patches/*

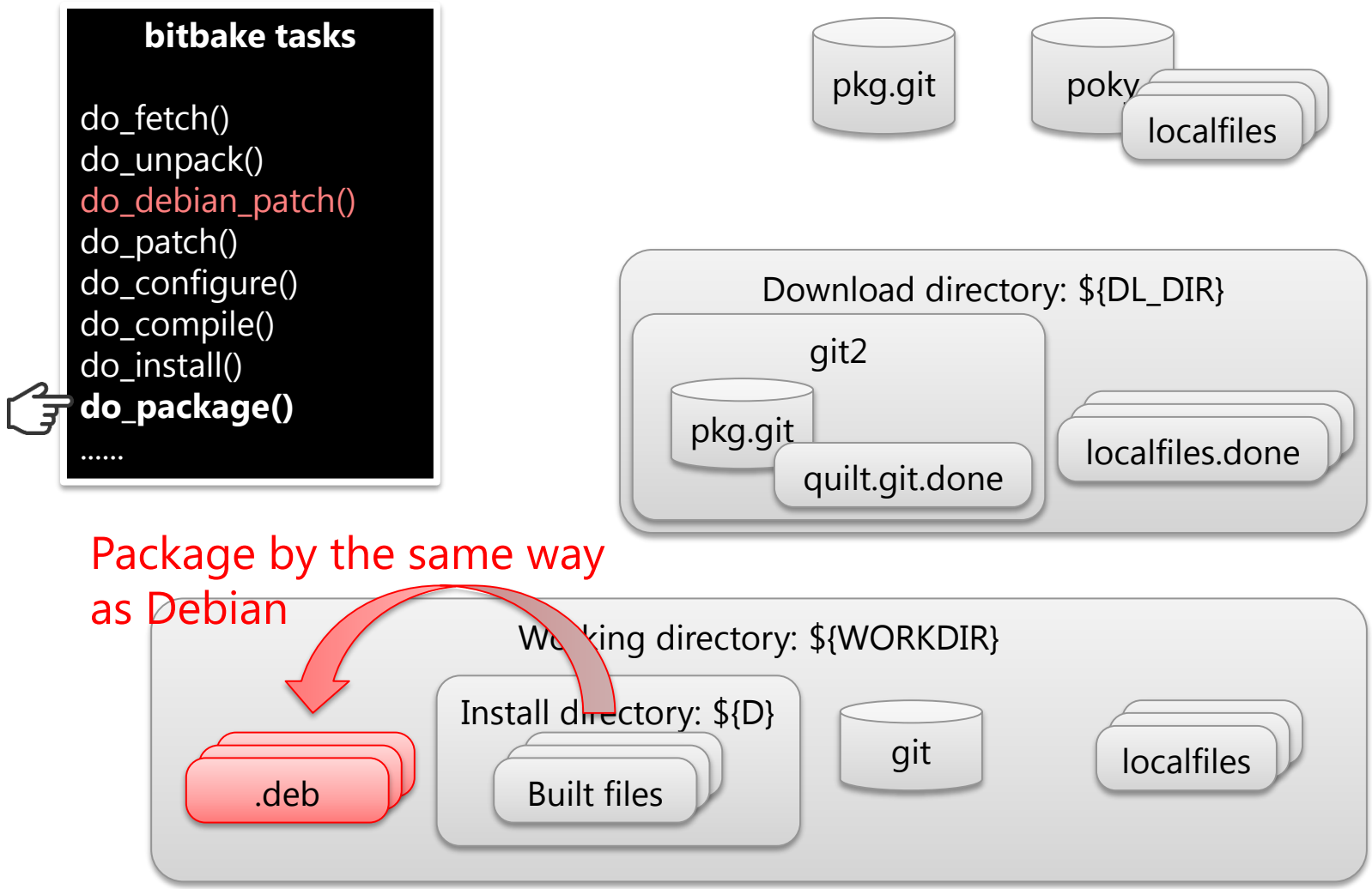
Build flow


bitbake tasks
do_fetch()
do_unpack()
do_debian_patch()
do_patch()
do_configure()
do_compile()
do_install()
do_package()
.....
pkg.git
poky
localfiles
Download directory: ${DL_DIR}
git2
pkg.git
quilt.git.done
localfiles.done
Apply patches for supporting cross-build
Working directory: ${WORKDIR}
git
localfiles

Build flow


bitbake tasks
do_fetch()
do_unpack()
do_debian_patch()
do_patch()
do_configure()
do_compile()
do_install()
do_package()
.....
pkg.git
poky
localfiles
Download directory: ${DL_DIR}
git2
pkg.git
quilt.git.done
localfiles.done
Install into the same paths as Debian
Configure & compile with same options as Debian
Install directory: ${D}
Built files
git
localfiles

Build flow


bitbake tasks
do_fetch()
do_unpack()
do_debian_patch()
do_patch()
do_configure()
do_compile()
do_install()
do_package()
.....
pkg.git
poky
localfiles
Download directory: ${DL_DIR}
git2
pkg.git
quilt.git.done
localfiles.done
Package by the same way as Debian
Working directory: ${WORKDIR}
Install directory: ${D}
Built files
.deb
git
localfiles

New features since October 2015

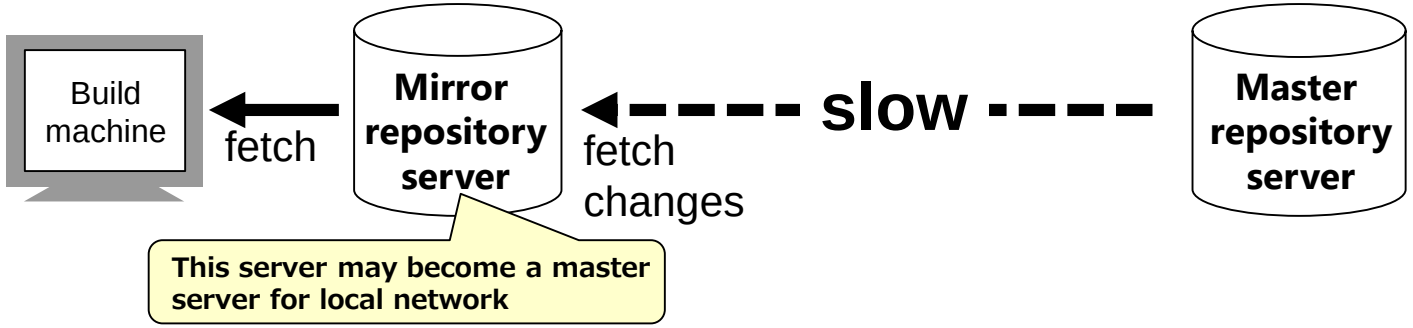
- **Follow the Poky upstream development**
  - Used be Daisy, now Jethro and master
- **Supported init systems**
  - Default init system is still busybox init
  - Added systemd support (Experimental)

- **Build an rootfs with LTP installed**
  - Using the LTP from upstream
- **Some useful features**
  - Git repository mirror server (Experimental)
  - TAG based source code fetch and build
  - Generate a summary information for image

Git repository mirror server

- **Issue**
  - Source fetch is really slow if network connection speed is slow

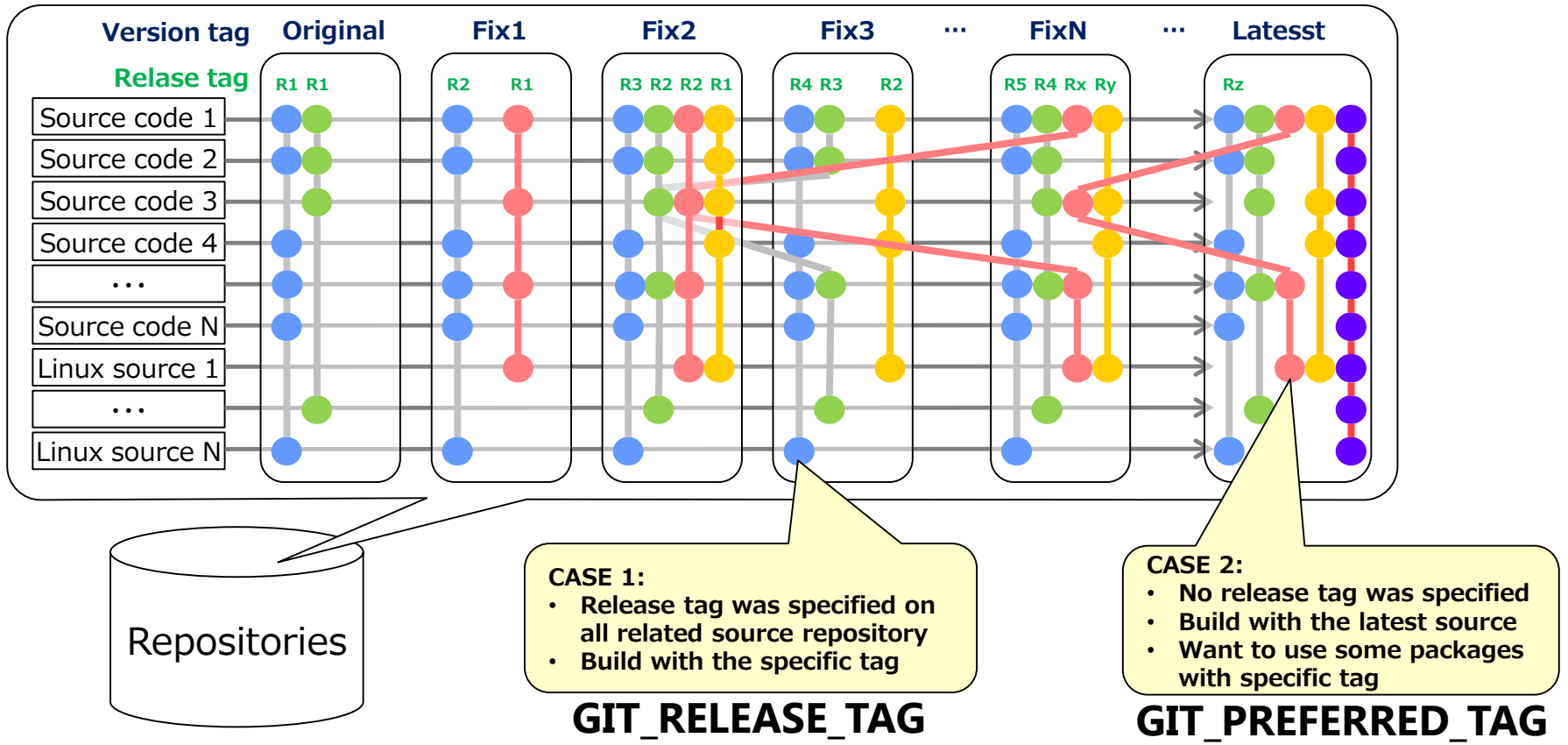
- **Solution**
  - Create a mirror server in local network environment
  - Initial time, it takes long to create mirror
  - Once it is created, fetch changes only



- **Create a docker image for Git repository server**
  - meta-debian-docker
  - Create a local repository server
  - Fecth update from master repository server

Tag based source code fetch and build

- **What purpose for?**
  - CASE 1: Re-create an specific image with same source code
  - CASE 2: Using a specific source version for one or more packages

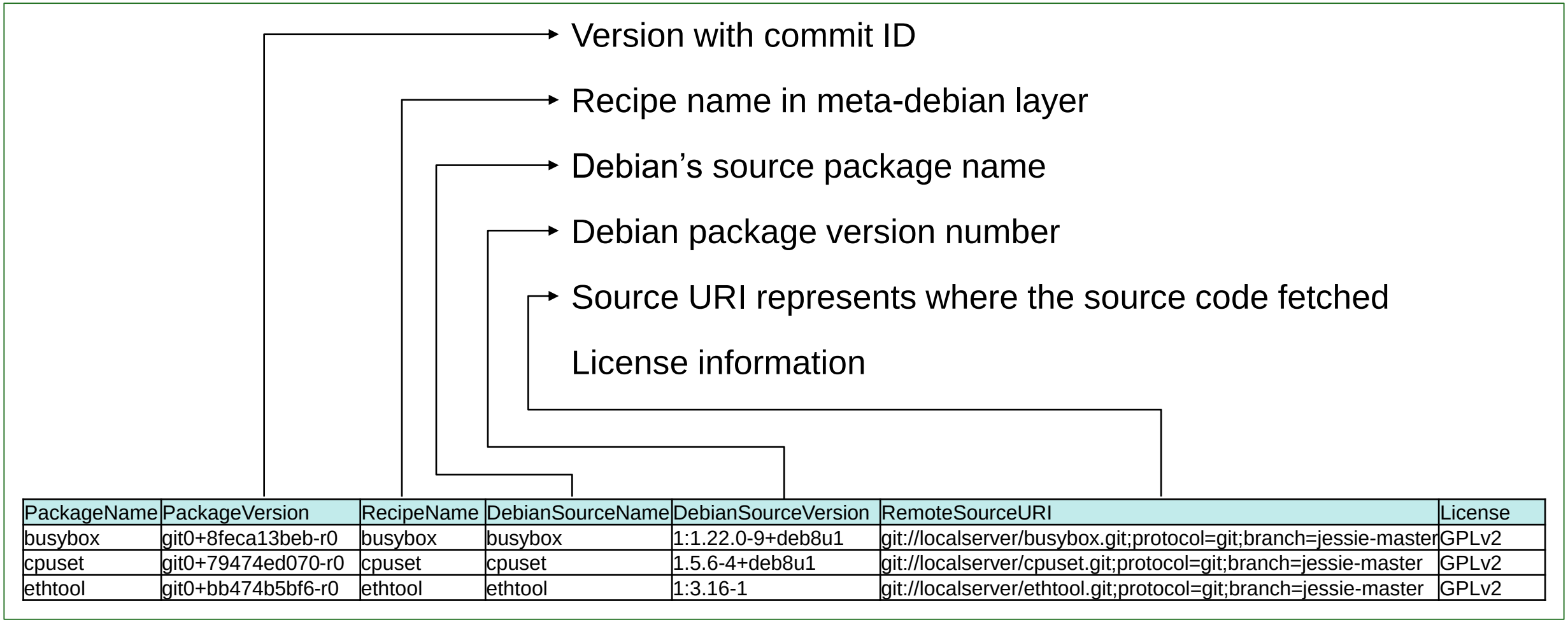


Tag based source code fetch and build

- **GIT_REBUILD_TAG:**
  - Assumption
    - Specified tag has been specified on all related repositories
  - Behavior
    - Rebuild with specified tag for each package
    - If the tag does not available on any repository, the build will fail
- **GIT_PREFERRED_TAG:**
  - Assumption
    - Specified tag has been specified on all or a **part of** related package
  - Behavior
    - Rebuild with specific tag, if it exist
    - If some repository does not have the specified tag, poky use the latest commit for the repository

Generate a summary information for images

- **Set the following configuration to generate information**
  - INHERIT += "summary"



PackageName	PackageVersion	RecipeName	DebianSourceName	DebianSourceVersion	RemoteSourceURI	License
busybox	git0+8feca13beb-r0	busybox	busybox	1:1.22.0-9+deb8u1	git://localserver/busybox.git;protocol=git;branch=jessie-master	GPLv2
cpuset	git0+79474ed070-r0	cpuset	cpuset	1.5.6-4+deb8u1	git://localserver/cpuset.git;protocol=git;branch=jessie-master	GPLv2
ethtool	git0+bb474b5bf6-r0	ethtool	ethtool	1:3.16-1	git://localserver/ethtool.git;protocol=git;branch=jessie-master	GPLv2

TODO

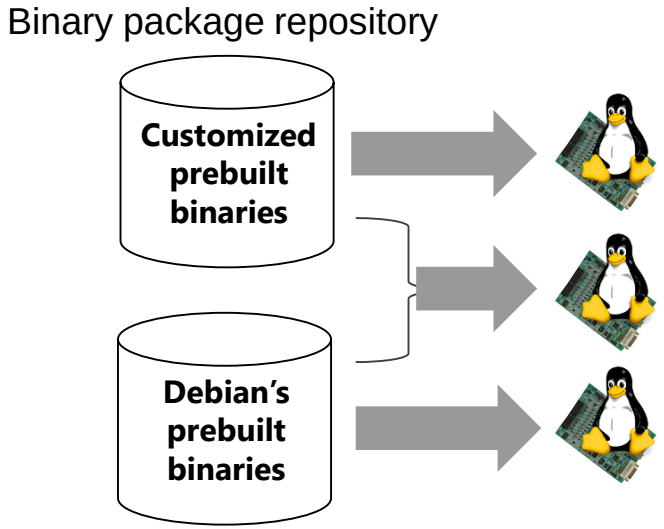
- **BSP**
  - Support more development boards (on going)
- **Recipe**
  - Add meta-zenbu (zenbu means “all” in Japanese)
    - Build with all packages included for build test
  - Add package-manager (especially dpkg or apt)
    - Install prebuild packages
  - Add a (semi-)automated recipe generator from Debian rules file
- **Test**
  - Integrate with Fuego

Things to be considerd to use prebuild package

- **Meta-debian layer aims to create fully customized environment based on Debian sources**

- **Issue**
  - Need to build from scratch and it takes long time
  - Partial update is difficult

- **Possible solutions**
  - Partially use pre-build binary
    - Create a cache storage to put all prebuild binaries
    - From Debian packages
      - Isar (https://github.com/ilbers/isar) enables to do it
    - Mix the both binaries into one (may cause another issue)



Current status

- **Supported CPUs**
  - x86 32bit and x86 64bit
  - ARM
  - PowerPC
- **Kernel**
  - LTSI with RT_PREEMPT patch set
- **User land**
  - busybox-based minimal system is still default
    - Also can be create bash-based minimal system
  - May work a window system
  - Number of available packages: around 400
    - Recipe implementation is still ongoing based on request

Current development status

	Oct 2015	Now
Poky version	Daisy	Jethro or master
Base Debian version for source code	Debian 8 (Jessie)	Debian 8 (Jessie)
Kernel	LTSI 3.10 and 3.14 + RT patch	3.10, 3.14, 4.1-LTSI + RT
Distribution Support	Debian 8 (Jessie)	Debian 8 (Jessie) / Ubuntu
Status	Under development	A bit stable
Number of packages	Approx. 200	Approx. 400
BSPs	QEMU(X86, X86_64, ARM, PowerPC) RasberryPi MinnowBoard	+ (planned) Dragonboard Intel Edison board

Conclusions

- **What is Shared Embedded Linux distribution**
  - Share the work of maintaining long-term support for an embedded distribution, by leveraging the work of the Debian project
    - Metadata for building embedded Linux systems using Debian source packages
    - Implemented as an independent layer of OpenEmbedded-Core
- **meta-debian is intended to provide**
  - Wide embedded CPU support
  - Stability
  - Long-term support
  - Fully customizable Linux

Please give us feedback

- **E-mail**
  - yoshitake.kobayashi@toshiba.co.jp
  - kazuhiro3.hayashi@toshiba.co.jp
- **Repository**
  - https://github.com/meta-debian/meta-debian.git
  - https://github.com/meta-debian/meta-debian-docker.git

Questions?

Thank you

How to create recipes (Sample: zlib)

meta-debian/recipe-debian/zlib/zlib_debian.bb

```
PR = "r0"
inherit debian-package

LICENSE = "Zlib"
LIC_FILES_CHKSUM = ¥
"file://zlib.h;beginline=4;endline=23;md5=fde612df1e5933c428b73844a0c494fd"

SRC_URI += "file://remove.ldconfig.call.patch"

do_configure() {
        ./configure --shared --prefix=${prefix} --libdir=${libdir}
}
do_compile () {
        oe_runmake
}
do_install() {
        oe_runmake DESTDIR=${D} install
}
do_install_append_class-target() {
        mkdir -p ${D}/${base_libdir}
        mv ${D}/${libdir}/libz.so.* ${D}/${base_libdir}
        tmp=`readlink ${D}/${libdir}/libz.so`
        ln -sf ../../${base_libdir}/$tmp ${D}/${libdir}/libz.so
}

DEBIANNAME_${PN}-dbg       = "${PN}1g-dbg"
DEBIANNAME_${PN}-staticdev = "${PN}1g-staticdev"
DEBIANNAME_${PN}-dev       = "${PN}1g-dev"
DEBIANNAME_${PN}-doc       = "${PN}1g-doc"
DEBIANNAME_${PN}           = "${PN}1g"
```

Step1: Add recipe revision

```
PR = "r0"
...

LICENSE = "Zlib"
LIC_FILES_CHKSUM
"file://zlib.h;b                                              8b73844a0c494fd"

SRC_URI += "file://remove.ldconfig.call.patch"

do_configure() {
        ./configure --shared --prefix=${prefix} --libdir=${libdir}
}
do_compile () {
        oe_runmake
}
do_install() {
        oe_runmake DESTDIR=${D} install
}
do_install_append_class-target() {
        mkdir -p ${D}/${base_libdir}
        mv ${D}/${libdir}/libz.so.* ${D}/${base_libdir}
        tmp=`readlink ${D}/${libdir}/libz.so`
        ln -sf ../../${base_libdir}/$tmp ${D}/${libdir}/libz.so
}

DEBIANNAME_${PN}-dbg       = "${PN}1g-dbg"
DEBIANNAME_${PN}-staticdev = "${PN}1g-staticdev"
DEBIANNAME_${PN}-dev       = "${PN}1g-dev"
DEBIANNAME_${PN}-doc       = "${PN}1g-doc"
DEBIANNAME_${PN}           = "${PN}1g"
```

- Define recipe revision: ${PR}
- Increment every update

Step2: Inherit debian-package.bbclass

```
inherit debian-package

LICENSE = "Zlib"
LIC_FILES_CHKSUM =
"file://zlib.h;beginl

SRC_URI += "file://remo

do_configure() {
        ./configure --shared --prefix=${prefix} --libdir=${libdir}
}
do_compile () {
        oe_runmake
}
do_install() {
        oe_runmake DESTDIR=${D} install
}
do_install_append_class-target() {
        mkdir -p ${D}/${base_libdir}
        mv ${D}/${libdir}/libz.so.* ${D}/${base_libdir}
        tmp=`readlink ${D}/${libdir}/libz.so`
        ln -sf ../../${base_libdir}/$tmp ${D}/${libdir}/libz.so
}

DEBIANNAME_${PN}-dbg       = "${PN}1g-dbg"
DEBIANNAME_${PN}-staticdev = "${PN}1g-staticdev"
DEBIANNAME_${PN}-dev       = "${PN}1g-dev"
DEBIANNAME_${PN}-doc       = "${PN}1g-doc"
DEBIANNAME_${PN}           = "${PN}1g"
```

Debian based

- Setup Debian source package
  - Define SRC_URI
  - Apply Debian's patches (do_debian_patch)

Step3: Add license information

```
PR = "r0"
inherit debian-package

LICENSE = "Zlib"
LIC_FILES_CHKSUM = ¥
"file://zlib.h;beginline=4;endline=23;md5=fde612df1e5933c428b73844a0c494fd"

SRC_URI += "fil://re

do_configure() {
        ./configu
}
do_compile () {
        oe_runmake
}
do_install() {
        oe_runmake
}
do_install_append_
        mkdir -p ${
        mv ${D}/${lib
        tmp=`readlink ${D}/${libdir}/libz.so`
        ln -sf ../../${base_libdir}/$tmp ${D}/${libdir}/libz.so
}

DEBIANNAME_${PN}-dbg       = "${PN}1g-dbg"
DEBIANNAME_${PN}-staticdev = "${PN}1g-staticdev"
DEBIANNAME_${PN}-dev       = "${PN}1g-dev"
DEBIANNAME_${PN}-doc       = "${PN}1g-doc"
DEBIANNAME_${PN}           = "${PN}1g"
```

- LICENSE: License name
  - Common license names are found in meta/files/common-licenses
- LIC_FILES_CHKSUM: Checksum of the license text
  - Usually found in COPYING, LICENSE, or header of source files (.c, .h)

Step4: Append patches

```
PR = "r0"
inherit debian-package

LICENSE = "Zlib"
LIC_FILES_CHKSUM = ¥
"file://zlib.h;beginline=4;endline=23;md5=fde612df1e5933c428b73844a0c494fd"

SRC_URI += "file://remove.ldconfig.call.patch"

do_co figure
}
do_compile
        oe_
}
do_install(
        oe_runmake DESTDIR=${D} install
}
do_install_append_class-target() {
        mkdir -p ${D}/${base_libdir}
        mv ${D}/${libdir}/libz.so.* ${D}/${base_libdir}
        tmp=`readlink ${D}/${libdir}/libz.so`
        ln -sf ../../${base_libdir}/$tmp ${D}/${libdir}/libz.so
}

DEBIANNAME_${PN}-dbg       = "${PN}1g-dbg"
DEBIANNAME_${PN}-staticdev = "${PN}1g-staticdev"
DEBIANNAME_${PN}-dev       = "${PN}1g-dev"
DEBIANNAME_${PN}-doc       = "${PN}1g-doc"
DEBIANNAME_${PN}           = "${PN}1g"
```

OE-Core based

- Add patches into SRC_URI
  - Necessary for being built in cross-compile environment
  - Copied from OE-Core (or create it from scratch)

Step5: Define configure options

```
PR = "r0"
inherit debian-package

LICENSE = "Zlib"
LIC_FILES_CHKSUM = ¥
"file://zlib.h;beginline=4;endline=23;md5=fde612df1e5933c428b73844a0c494fd"

SRC_URI += "file://remove.ldconfig.call.patch"

do_configure() {
        ./configure --shared --prefix=${prefix} --libdir=${libdir}
}
do_compile () {
        oe
}
do_instal
        oe
}
do_install
        mk
        mv ${D}/${libdir}/libz.so.* ${D}/${base_libdir}
        tmp=`readlink ${D}/${libdir}/libz.so`
        ln -sf ../../${base_libdir}/$tmp ${D}/${libdir}/libz.so
}

DEBIANNAME_${PN}-dbg       = "${PN}1g-dbg"
DEBIANNAME_${PN}-staticdev = "${PN}1g-staticdev"
DEBIANNAME_${PN}-dev       = "${PN}1g-dev"
DEBIANNAME_${PN}-doc       = "${PN}1g-doc"
DEBIANNAME_${PN}           = "${PN}1g"
```

Debian based

- Define configure commands
  - The same options as debian/rules
  - Some features should be disabled for embedded

Step6: Define compile and install commands

```
PR = "r0"
inherit debian-package

LICENSE = "Zlib"
LIC_FILES_CHKSUM = ¥
"file://zlib.h;beginline=4;endline=23;md5=fde612df1e5933c428b73844a0c494fd"

SRC_URI += "file://remove.ldconfig.call.patch"

do_configure() {
        ./configure --shared --prefix=${prefix} --libdir=${libdir}
}
do_compile () {
        oe_runmake
}
do_install() {
        oe_runmake DESTDIR=${D} install
}
do_install_append_class-target() {
        mkdir -p ${D}/
        mv ${D}/${li
        tmp=`readlink
        ln -sf ../../$
}

DEBIANNAME_${PN}-dbg       = "${PN}1g-dbg"
DEBIANNAME_${PN}-staticdev = "${PN}1g-staticdev"
DEBIANNAME_${PN}-dev       = "${PN}1g-dev"
DEBIANNAME_${PN}-doc       = "${PN}1g-doc"
DEBIANNAME_${PN}           = "${PN}1g"
```

- Define compile & install commands
  - autotools.bbclass often replaces them

Additional Steps: Change library paths

```
PR = "r0"
inherit debian-package

LICENSE = "Zlib"
LIC_FILES_CHKSUM = ¥
"file://zlib.h;beginline=4;endline=23;md5=fde612df1e5933c428b73844a0c494fd"

SRC_URI += "file://remove.ldconfig.call.patch"

do_configure() {
        ./configure --shared --prefix=${prefix} --libdir=${libdir}
}
do_compile () {
        oe_r
}
do_install()
        d_i
}
do_install_append_class-target() {
        mkdir -p ${D}/${base_libdir}
        mv ${D}/${libdir}/libz.so.* ${D}/${base_libdir}
        tmp=`readlink ${D}/${libdir}/libz.so`
        ln -sf ../../${base_libdir}/$tmp ${D}/${libdir}/libz.so
}

DEBIANNAME_${PN}-dbg       = "${PN}1g-dbg"
DEBIANNAME_${PN}-staticdev = "${PN}1g-staticdev"
DEBIANNAME_${PN}-dev       = "${PN}1g-dev"
DEBIANNAME_${PN}-doc       = "${PN}1g-doc"
DEBIANNAME_${PN}           = "${PN}1g"
```

Debian based

Move run-time libraries to the same directory as Debian

Additional Steps: Change package name

```
PR = "r0"
inherit debian-package

LICENSE = "Zlib"
LIC_FILES_CHKSUM = ¥
"file://zlib.h;beginline=4;endline=23;md5=fde612df1e5933c428b73844a0c494fd"

SRC_URI += "file://remove.ldconfig.call.patch"

do_configure() {
        ./configure --shared --prefix=${prefix} --libdir=${libdir}
}
do_compile () {
        oe_runmake
}
do_install() {
        oe_runmake DESTDIR=${D} install
}
do_install_app
        mkdir
        mv ${D
        tmp=`
        ln
}

DEBIANNAME_${PN}-dbg       = "${PN}1g-dbg"
DEBIANNAME_${PN}-staticdev = "${PN}1g-staticdev"
DEBIANNAME_${PN}-dev       = "${PN}1g-dev"
DEBIANNAME_${PN}-doc       = "${PN}1g-doc"
DEBIANNAME_${PN}           = "${PN}1g"
```

Debian based

- Change the default binary package name to Debian's
- "libz" => "zlib1g"

Build results (zlib packages)

Debian 8.0 jessie

zlib1g
- /lib/i386-linux-gnu/libz.so.1
- /lib/i386-linux-gnu/libz.so.1.2.8
- /usr/share/doc/zlib1g/...

zlib1g-dev
- /usr/include/i386-linux-gnu/zconf.h
- /usr/include/zlib.h
- /usr/lib/i386-linux-gnu/libz.so
- /usr/lib/i386-linux-gnu/pkgconfig/zlib.pc
- /usr/share/doc/...
- /usr/share/man/...
- /usr/lib/i386-linux-gnu/libz.a

zlib1g-dbg

zlib1g-udeb

lib32z1* | lib64z1* | libn32z1*

meta-debian layer

zlib1g
- /lib/libz.so.1
- /lib/libz.so.1.2.8

zlib1g-dev
- /usr/include/zconf.h
- /usr/include/zlib.h
- /usr/lib/libz.so
- /usr/lib/pkgconfig/zlib.pc

zlib1g-doc
- /usr/share/man/...

zlib1g-staticdev
- /usr/lib/libz.a

zlib1g-dbg

Ignore non-essential files